Supplementary Data

March 15, 2019

1 S.0 Evaluation of computational fluid dynamics and lumped-parameter modelling approaches for planktonic phototrophic growth: Supplementary data

This case file compares the solution of the standard PAM solver as published by Puyol et al, and that of the recently developed CFD solution, namely the combination of the `photoBio` libraries and `pamFoam` solver. There will be three different comparisons with the complete Eulerian solution: 1. Fixed radiative field equal to that of incident irradiance (in this case, we have imposed 30 W/m2 @ 850 nm 2. Fixed radiative field at 8.7 W/m2, the radiative field is constant at the value of the half-saturation constant. 2. Dynamic radiative field based on U.water field streamlines in the radiative field at an average fixed

There is an hypothesis that flat plate PBRs can be approximated to CSTRs if one does some appropriate investigation and understands the coupling between the radiative field and the biokinetics. This only applies to the the flat plate reactor

```
In [1]: from IPython.display import HTML
```

```
In [2]: HTML('''<script>
        code_show=true;
        function code_toggle() {
         if (code_show){
         $('div.input').hide();
         } else {
         $('div.input').show();
         }
         code_show = !code_show
        }
        $( document ).ready(code_toggle);
        </script>
        The raw code for this IPython notebook is by default hidden for easier reading.
        To toggle on/off the raw code, click <a href="javascript:code_toggle()">here</a>.''')
```

```
Out[2]: <IPython.core.display.HTML object>
```

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

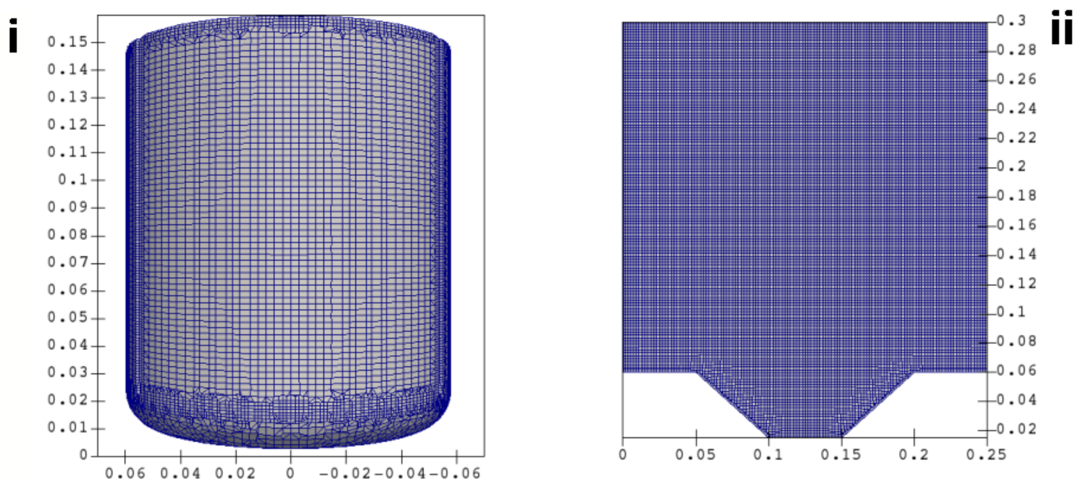
```
from scipy.integrate import odeint
%matplotlib inline
plt.style.use('seaborn-whitegrid')
```

2 S.1 Mesh Generation

Below is are the meshes for both the stirred vessel reactor (CSTR) and the flat plate reactor (FPR). Both meshes were done using cfMesh, and they both consist of 300k cells.

```
In [4]: from IPython.core.display import Image
        Image(filename='graphs/meshes.png')
```

```
Out[4]:
```



3 S.2 Initial Conditions for the simulations

State	Initial Condition	Units	Meaning
S S	0.2	gCOD/L	soluble organic substrate excluding acetic acid
S AC	0.5	gCOD/L	acetic acid substrate
S I	0.15	gCOD/L	inert soluble matter
S IN	0.045	gN/L	ammonium concentration as nitrogen
S IP	0.00815	gP/L	phosphorus concentration as phosphate
X PB	0.5	gCOD/L	PPB biomass concentration as COD
X S	0.3	gCOD/L	biodegradable organic particulates
X I	0.2	gCOD/L	inert particulate matter

In [5]:
```python
# Define the PAM solver for the ODE systems

def monod(I, Ks):
    return I / (Ks + I)


def pam_batch(y, t, u):
    """ Definition of the system of differential equations to integrate."""

    # Unpack the states
    # Assign variables for convenience of notation
    SS = y[0]
    SAC  = y[1]
    SIC = y[2]
    SH2 = y[3]
    SIN = y[4]
    SIP = y[5]
    SI = y[6]
    XPB = y[7]
    XS = y[8]
    XI = y[9]
    #SCAT = 0.001+y[2]*0.3

    # Define parameters of the model (basis days)
    fSSXS = 1.6382408312061000E-01
    fSAXS = 1.166839250294608E-01
    fICXS = 1.3039707398869100E-06
    fH2XS = 8.4424680871970500E-02
    fINXS = 0.011622
    fIPXS = 0.002835
    fSIXS = 0.1518209
    fXIXS = 0.4330922

    fICPHAC = 6.44841269841271e-6     # molHCO3-C/mgCOD
    fICPHSS = -1.242761443579e-6      # molHCO3-C/mgCOD

    fACCH = 0.6691          # mgCOD/mgCOD
    fH2CH = 1.0 - fACCH     # mgCOD/mgCOD

    fICAU = 1.0           # molHCO3-C/molHCO3-C
    fH2AU = 40320.0       # mgCOD/molHCO3-C

    fNB = 0.086           # mgNH3-N/mgCOD
    fPB = 0.015           # mgPO4-P/mgCOD

    kHYD = 7.09e-2        # d-1
    kDEC = 9.00e-2        # d-1
    kMAC = 2.375          # d-1
```

```
kMPH = 1.435          # d-1
kMCH = 7.386e-2       # d-1
kMIC = 6.0e-6         # mol HCO3-C/mgCOD/d

KSS = 0.524235146324262      # mg COD/L
KSAC = 20.222562            # mg COD/L
KSIC = 4.2e-4               # mol HCO3-C/L
KSIN = 0.02                 # mgNH3-N/L
KSIP = 0.081                # mgPO4-P/L
KIFA = 7850.0               # mg NH4-N/L
KSH2 = 1.0                  # mg COD/L

YPBPH = 1.0                 # mgCOD/mgCOD
YPBCH = 0.680705638006362   # mgCOD/mgCOD
YPBAU = 40320.00            # mgCOD/molHCO3-C

KSE = 8.76                  # W/m2
SE = u                      # W/m2
fICDEC = -1.98412698412703E-07 # mol C-HCO3 / mgCOD
fINDEC = 0.058                 # mgNH3-N / mgCOD
fIPDEC = 0.01                  # mgPO4-P / mgCOD

# Constitutive equations
rHYD = kHYD * XS
rDEC = kDEC * XPB

IIN = SIN/(KSIN + SIN)
IIP = SIP/(KSIP + SIP)
IFA = KIFA/(KIFA + SIN)
IE = monod(SE, 8.76)
ICS = SAC / (SS + SAC)      # ACT inhibition due to SS
ICAC = SS / (SS + SAC)      # PHT inhibition due to SAC


rACT = kMAC * XPB * IFA * IIN * IIP * IE * (SAC/(KSAC + SAC)) * ICS
rPHT = kMPH * XPB * IFA * IIN * IIP * IE * (SS/(KSAC + SAC)) * ICAC
rCHE = kMCH * XPB * IFA * IIN * IIP * SS/(KSS + SS)
rAUT = kMIC * XPB * IFA * IIN * IIP * IE * SIC/(SIC + KSIC)*SH2/(SH2 + KSH2);


n = len(y)       # 10: number of states
dydt = np.empty((n))

dydt[0] = fSSXS * rHYD \
        - rPHT \
        - rCHE

dydt[1] = fSAXS * rHYD \
```

```
            - rACT \
            + (1 - YPBCH)*fACCH*rCHE

    dydt[2] = fICXS * rHYD \
            + fICPHAC * rACT \
            + fICPHSS * rPHT \
            - fICAU * rAUT \
            + fICDEC * rDEC

    dydt[3] = fH2XS * rHYD \
            + (1 - YPBCH) * fH2CH * rCHE \
            - fH2AU * rAUT

    dydt[4] = fINXS*rHYD \
            - fNB*YPBPH*rACT \
            - fNB*YPBPH*rPHT \
            + fINDEC*rDEC \
            - fNB*YPBCH*rCHE \
            - fNB*YPBAU*rAUT


    dydt[5] = fIPXS*rHYD \
            - fPB*YPBPH*rACT \
            - fPB*YPBPH*rPHT \
            + fIPDEC*rDEC \
            - fPB*YPBCH*rCHE \
            - fPB*YPBAU*rAUT


    dydt[6] = fSIXS*rHYD

    dydt[7] = YPBPH*rACT \
            + YPBPH*rPHT \
            - rDEC \
            + YPBCH*rCHE \
            + YPBAU*rAUT


    dydt[8] = rDEC - rHYD

    dydt[9] = fXIXS * rHYD

    return dydt
```

In [6]:
```
# 30 W/m2 uniform irradiance
#########################################################
# USER INPUT
num_steps = 800
```

```python
days = 120604/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45  # 45
SIP0 = 8.15   # 8.15
SI0 = 150.0
XPB0 = 500.0
XS0 = 300.0
XI0 = 100.0

##########################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

time_ode = np.linspace(0,days,num_steps)
#u_t = interp1d(time_input.squeeze(), full_G850.squeeze())

# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps - 1):
    t = [time_ode[step], time_ode[step+1]]

    # Get G850 for this step
    G850 = 30

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(G850,))

    y_inter = np.float64(soln[1])

    y_out.append(y_inter)

    y0 = y_inter.tolist()


y_out = np.asarray(y_out)
df_30 = pd.DataFrame(y_out)
df_30['SSout'] =  y_out[:,0]
df_30['SACout'] = y_out[:,1]
df_30['SICout'] = y_out[:,2]
df_30['SH2out'] = y_out[:,3]
df_30['SINout'] = y_out[:,4]
df_30['SIPout'] = y_out[:,5]
```

```
df_30['SIout'] = y_out[:,6]
df_30['XPBout'] = y_out[:,7]
df_30['XSout'] = y_out[:,8]
df_30['XIout'] = y_out[:,9]
df_30['time'] = time_ode
```

In [7]:
```
# Beer Lambert irradiance (ODE2)

#########################################################
# USER INPUT
num_steps = 800

days = 120604/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45  # 45
SIP0 = 8.15   # 8.15
SI0 = 150.0
XPB0 = 500.0
XS0 = 300.0
XI0 = 100.0

def beer_lambert(G0, AplusS, L):
        return G0*np.exp(-AplusS*L)

G850 = beer_lambert(30, 53+15, 0.0075)

##########################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

time_ode = np.linspace(0,days,num_steps)
#u_t = interp1d(time_input.squeeze(), full_G850.squeeze())

# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps - 1):
    t = [time_ode[step], time_ode[step+1]]

    # Get G850 for this step
    G850 = beer_lambert(30, 53, 0.015)

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(G850,))
```

```
        y_inter = np.float64(soln[1])

        y_out.append(y_inter)

        y0 = y_inter.tolist()


    y_out = np.asarray(y_out)
    df_10 = pd.DataFrame(y_out)
    df_10['SSout'] =  y_out[:,0]
    df_10['SACout'] = y_out[:,1]
    df_10['SICout'] = y_out[:,2]
    df_10['SH2out'] = y_out[:,3]
    df_10['SINout'] = y_out[:,4]
    df_10['SIPout'] = y_out[:,5]
    df_10['SIout'] = y_out[:,6]
    df_10['XPBout'] = y_out[:,7]
    df_10['XSout'] = y_out[:,8]
    df_10['XIout'] = y_out[:,9]
    df_10['time'] = time_ode
```

4 S.3 Solve transient radiative field problem using streamlines on U.water

We will first do the process for the flat plate multiphase bioreactor, then we can look at the stirred vessel.

```
In [8]: stream_flat = pd.read_csv("./data/streamlines1.csv")["G850"]
        streamtime_flat = pd.read_csv("./data/streamlines1.csv")["IntegrationTime"]

        # Check where time goes back to zero,
        # If the size of the resulting array is greater than a certain value, M,
        # then return that to a list of indices

        # This is for the flat plate reactor
        lst_flat = []
        M = 500
        j = 0
        starter = 30
        j = starter

        while starter < len(streamtime_flat)-2:
            while streamtime_flat[j] < streamtime_flat[j+1]:
                j = j + 1
            if j - starter > M:
                lst_flat.append([starter, j])
```

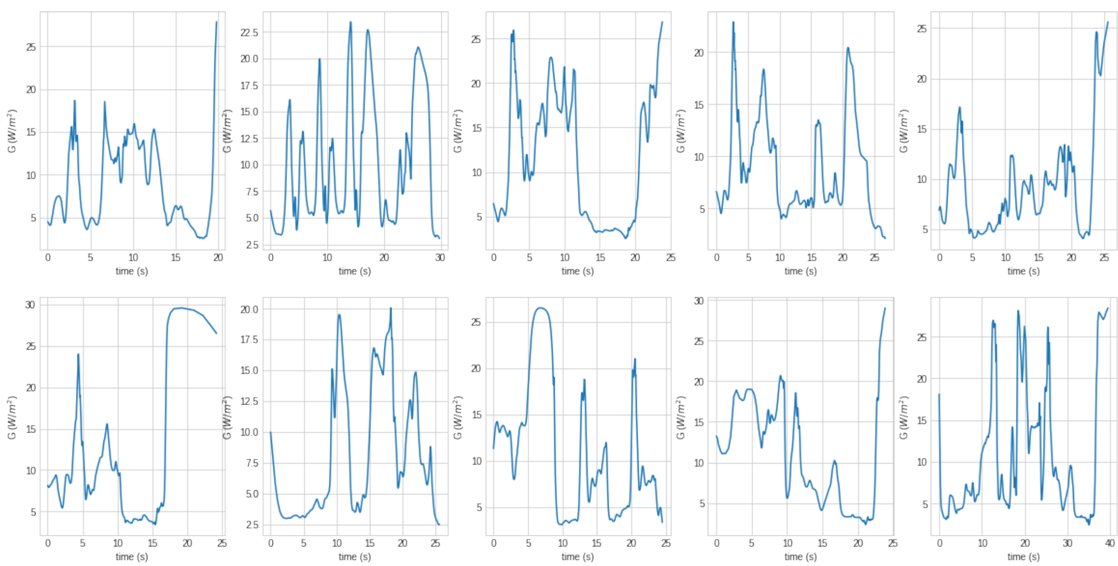
```
            starter = j + 1
            j = starter
        print(starter, j)
```

```
34647 34647
```

As we can see from the results above, there are 10 intervals which give us more than 500 datapoints each. These intervals will be used as a basis of potential candidates for the dynamic particles. Below, we can see the graphic of the streamlines as they experience the radiative field at 850 nm.

```
In [9]: plt.figure(figsize=(20,10))
        for i in range(len(lst_flat)):
            plt.subplot(2, 5, i+1)
            plt.plot(streamtime_flat[lst_flat[i][0]:lst_flat[i][1]], stream_flat[lst_flat[i][0
            plt.xlabel('time (s)')
            plt.ylabel('G ($W/m^2$)')

        lst_flat[9]
```

```
Out[9]: [14548, 15424]
```

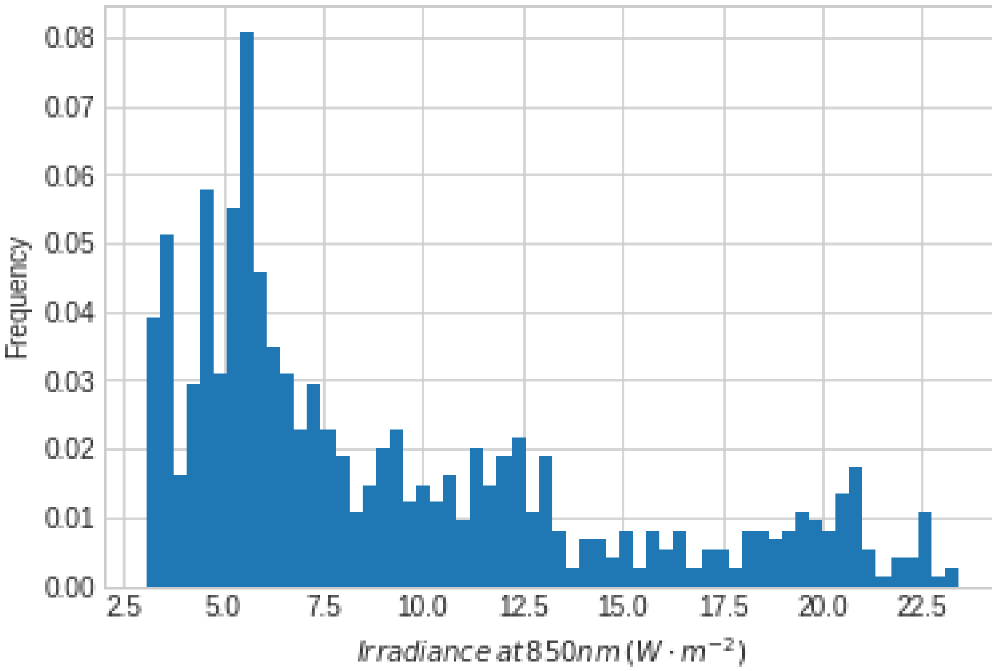


4.1 Determination of the probability law of the streamlines

We can see in the following graphic the probability distribution and the important parameters of the flat plate G distribution. The rate parameter, λ is 0.16, and the minimum value of G over the selected stream is 3.05 W/m2.

In [10]:
```
# Determination of Law

import scipy.stats as stats
part0 = stream_flat[lst_flat[1][0]:lst_flat[1][1]]
time0 = streamtime_flat[lst_flat[1][0]:lst_flat[1][1]]
meanG = np.log(stream_flat[lst_flat[1][0]:lst_flat[1][1]]).mean()
stdG = np.log(part0).std()
maxG = part0.max()

particle_flat = part0
minG_flat = particle_flat.min()
time_flat = time0

loc_flat,scale_flat = stats.expon.fit(particle_flat, floc=minG_flat)
x_flat = np.linspace(0,30)
dist_flat = stats.expon.pdf(x_flat, loc_flat, scale_flat)

weights_flat = np.ones_like(particle_flat)/float(len(particle_flat))

plt.hist(particle_flat, 60, weights=weights_flat)
plt.xlabel('${Irradiance\, at\, 850 nm\, (W\cdot m^{-2}})$')
plt.ylabel('Frequency')
plt.show()

print(loc_flat, scale_flat)
print(f'rate = {1/scale_flat}')
print(f'minG = {minG_flat}')


time0.index = pd.RangeIndex(len(time0.index))
part1 = np.concatenate(60*24*3*[part0])
time1 = np.concatenate(60*24*3*[time0])

# Create a deltaTime list
deltaTime = []
for i in range(len(time0)-1):
    deltaTime.append(time0[i+1] - time0[i])

time1 = [0]


# Turn deltaTime into a np.array
deltaTime = np.concatenate((60*24*3+4)*[np.array(deltaTime)])

for i in range(len(deltaTime)):
    time1.append(time1[-1]+deltaTime[i])

from scipy.interpolate import interp1d
```

```
time2 = np.asarray(time1)/86400
f = interp1d(time2, part1[0:len(time1)])
Nsteps = 800
```




```
3.0435 6.172651881720432
rate = 0.1620049241657998
minG = 3.0435
```

In [11]:
```
# ODE3 (dynamic input)

##########################################################
# USER INPUT
num_steps = 10000

days = 172800/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45   # 45
SIP0 = 8.15    # 8.15
SI0 = 150.0
XPB0 = 500.0
```

```python
XS0 = 300.0
XI0 = 100.0

############################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

# Interpolate the irradiance array

t_new = np.linspace(0, time2.max(), num_steps)
G850 = f(t_new)

# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps-1):
    t = [t_new[step], t_new[step+1]]

    # Get G850 for this step
    u = G850[step:step+1]

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(u,))

    y_inter = np.float64(soln[1])

    y_out.append(y_inter)

    y0 = y_inter.tolist()

y_out = np.asarray(y_out)
df_dyn = pd.DataFrame((y_out))

df_dyn['SSout'] =  y_out[:,0]
df_dyn['SACout'] = y_out[:,1]
df_dyn['SICout'] = y_out[:,2]
df_dyn['SH2out'] = y_out[:,3]
df_dyn['SINout'] = y_out[:,4]
df_dyn['SIPout'] = y_out[:,5]
df_dyn['SIout'] = y_out[:,6]
df_dyn['XPBout'] = y_out[:,7]
df_dyn['XSout'] = y_out[:,8]
df_dyn['XIout'] = y_out[:,9]
df_dyn['time'] = t_new
```

5 S.4 Importing of CFD Data

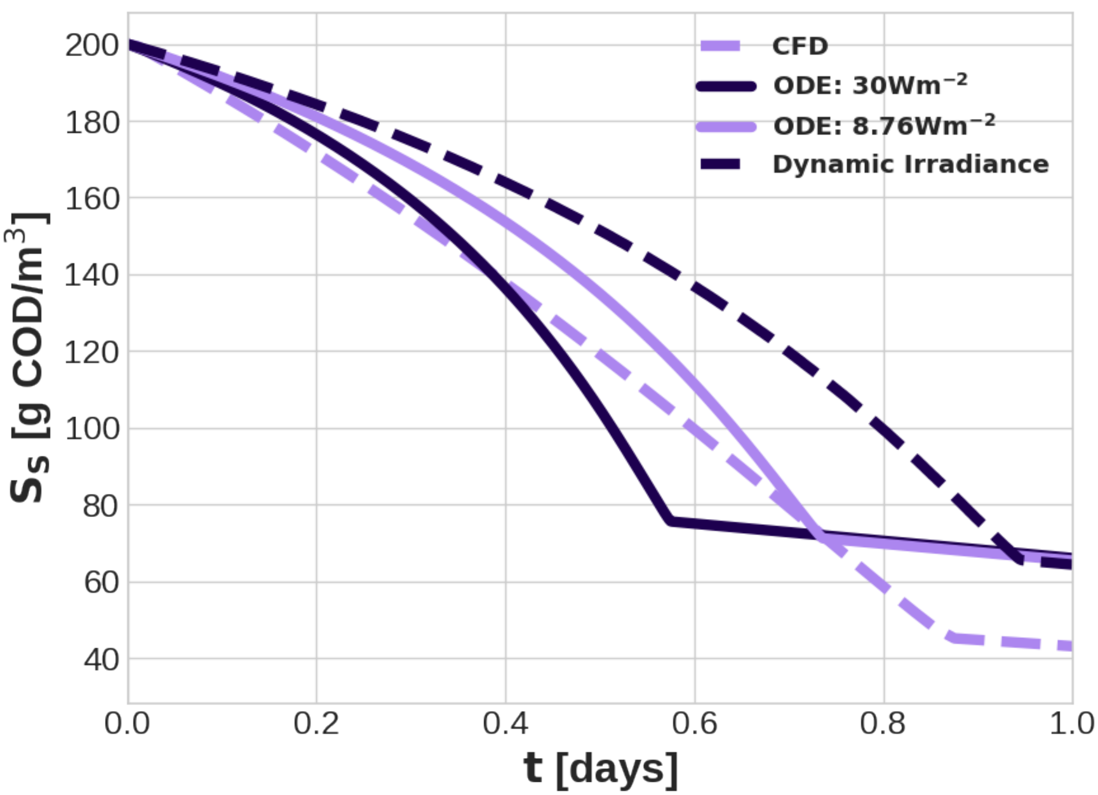
Here we impport the CFD results from the FPR and CR. Once the simulations are run on the ODE system, one must import the results from the no-flow solution from the OpenFOAM solver. In this case, the irradiance was assumed as equal throughout the entire domain at 30 $\mathtt{Wm}^{-2}$.

```
In [12]: import pandas as pd

         data = pd.read_csv("data/pamFieldData.dat",
                            sep='\t',
                            skiprows=range(3))
         data.columns = ["time",
                         "SS", "SAC", "SIC", "SH2", "SIN", "SIP", "SI",
                         "XPB", "XS", "XI", "G850", "magUWater"]

         cstr = pd.read_csv("data/stirredVesselAverages/volFieldValue.dat",
                             sep='\t',
                             skiprows=range(3))

         cstr.columns = ["time", "ss", "sac", "sin", "sip", "xpb", "xs", "xi",  "G850",]
```

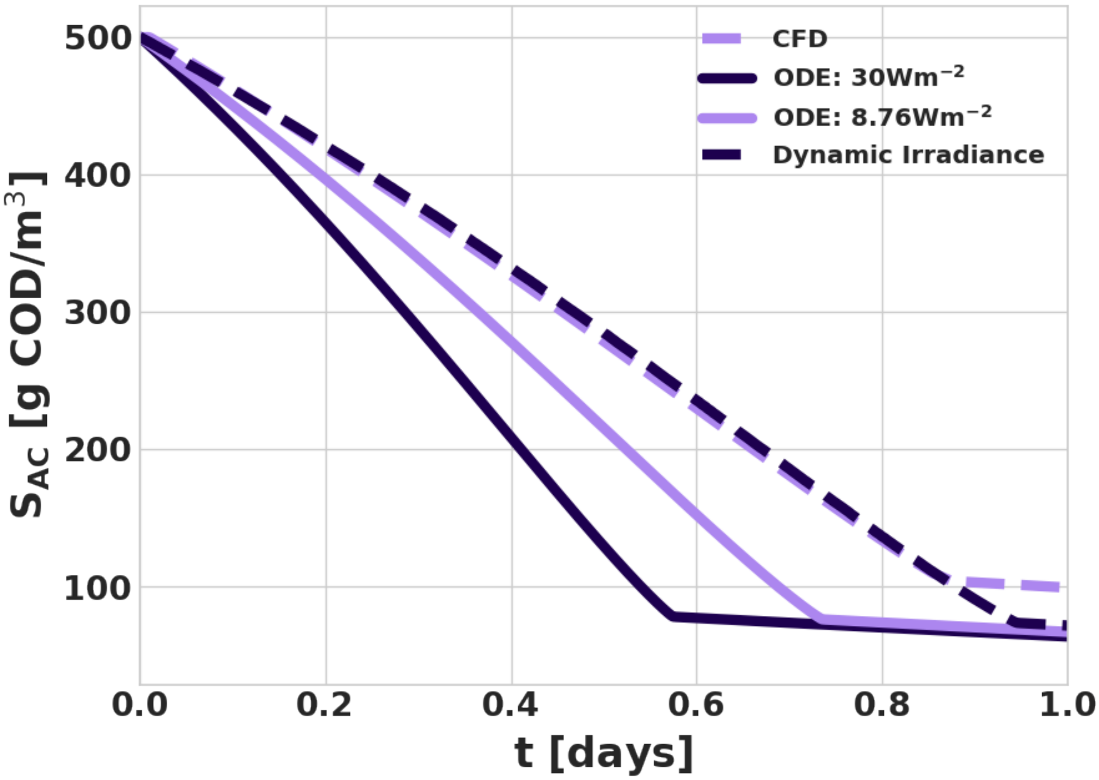
N.B. the medium and coarse solutions overlapped, with the generated graphics being available in the supplementary information directory.

```
In [13]: fig1 = plt.figure(figsize=(8,6), dpi=120)
         ax1 = fig1.add_subplot(111)

         ax1.tick_params(axis="both",
                         reset=False,
                         labelsize=16)
         font = {'family' : 'DejaVu Sans',
                 'weight' : 'bold',
                 'size'   : 14}

         plt.rc('font', **font)


         plt.plot(data["time"]/86400,
                  data["SS"]*1000,
                  "--",
                  label='CFD',
                  linewidth=5,
                  color="#ac86ef")

         plt.plot(df_30["time"],
                  df_30["SSout"],
                  label='ODE: 30$\mathbf{Wm^{-2}}$',
                  linewidth=5,
                  color="#1d004f")
```

```
plt.plot(df_10["time"],
         df_10["SSout"],
         label='ODE: 8.76$\mathbf{Wm^{-2}}$',
         linewidth=5,
         color="#ac86ef")

plt.plot(df_dyn["time"],
         df_dyn["SSout"],
         "--",
         label='Dynamic Irradiance',
         linewidth=5,
         color="#1d004f"
         )

plt.legend(fontsize=12)
plt.xlabel(r'$\mathbf{t}$ [days]', fontweight='bold', fontsize=20)
plt.ylabel(r'$\mathbf{S_S}$ [g COD/m$^3$]', fontweight='bold', fontsize=20)
plt.xlim([0, 1])

fig1.savefig("graphs/SS_evol.eps")
```

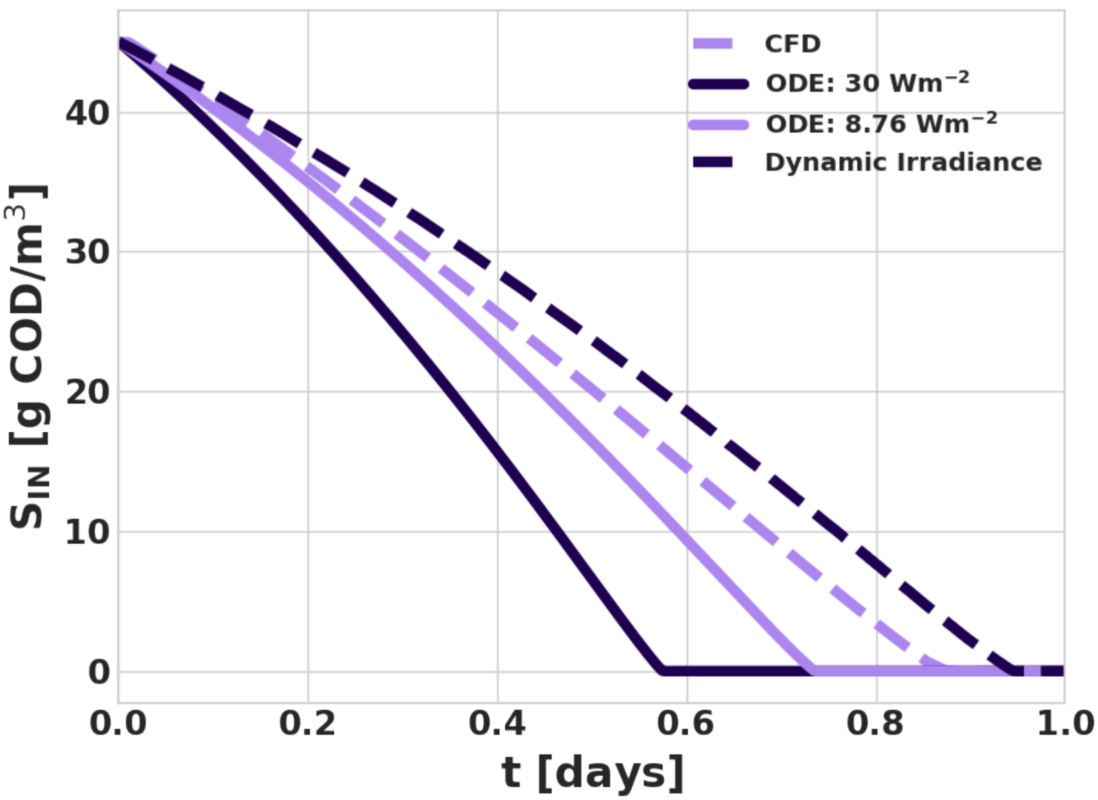


```
In [14]: fig2 = plt.figure(figsize=(8,6), dpi=120)
         ax1 = fig2.add_subplot(111)

         ax1.tick_params(axis="both",
                         reset=False,
                         labelsize=16)
         font = {'family' : 'DejaVu Sans',
                 'weight' : 'bold',
                 'size'   : 14}

         plt.rc('font', **font)

         plt.plot(data["time"]/86400,
                  data["SAC"]*1000,
                  "--",
                  label='CFD',
                  linewidth=5,
                  color="#ac86ef")

         plt.plot(df_30["time"],
                  df_30["SACout"],
                  label='ODE: 30$\mathbf{Wm^{-2}}$',
                  linewidth=5,
                  color="#1d004f")

         plt.plot(df_10["time"],
                  df_10["SACout"],
                  label='ODE: 8.76$\mathbf{Wm^{-2}}$',
                  linewidth=5,
                  color="#ac86ef")

         plt.plot(df_dyn["time"],
                  df_dyn["SACout"],
                  "--",
                  label='Dynamic Irradiance',
                  linewidth=5,
                  color="#1d004f")

         plt.legend(fontsize=12)
         plt.xlabel(r'$\mathbf{t}$ [days]', fontweight='bold', fontsize=20)
         plt.ylabel(r'$\mathbf{S_{AC}}$ [g COD/m$^3$]', fontweight='bold', fontsize=20)
         plt.xlim([0, 1])

         fig2.savefig("graphs/SAC_evol.eps")
```



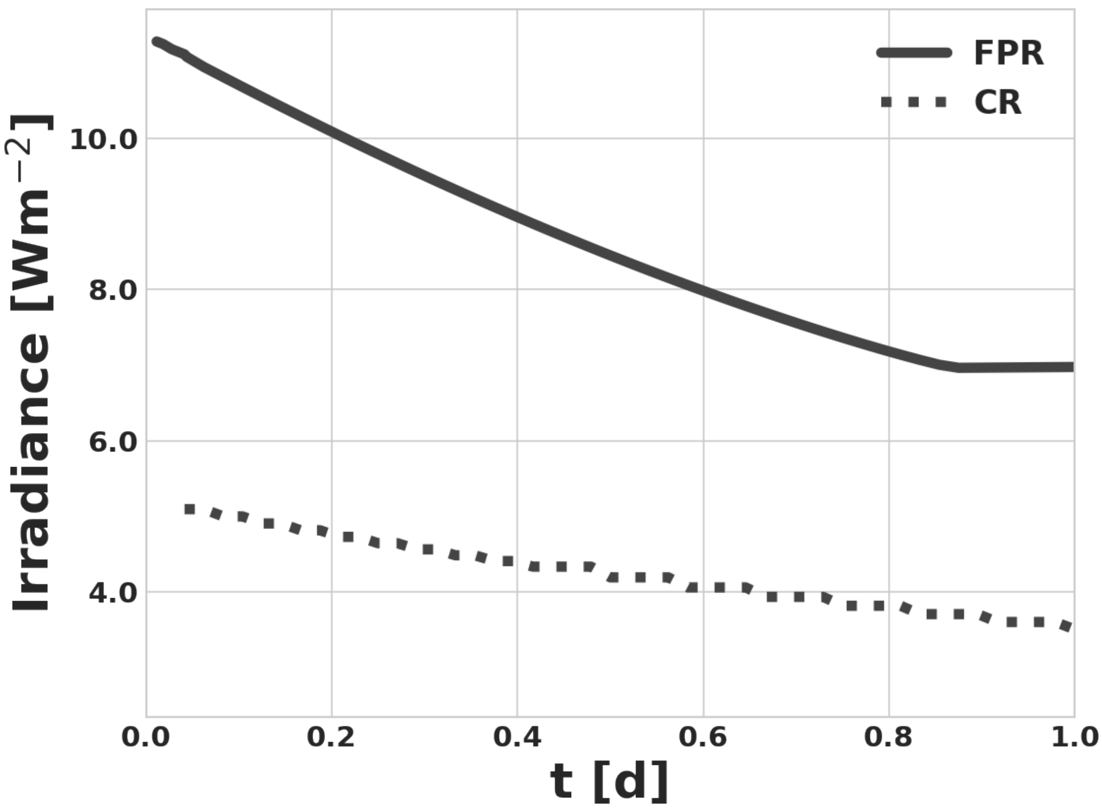
In [15]:
```
fig3 = plt.figure(figsize=(8,6), dpi=120)
ax1 = fig3.add_subplot(111)

ax1.tick_params(axis="both",
                reset=False,
                labelsize=16)
font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 14}

plt.rc('font', **font)



plt.plot(data["time"]/86400,
         data["SIN"]*1000,
         "--",
         label='CFD',
         linewidth=5,
         color="#ac86ef")

plt.plot(df_30["time"],
```

```
         df_30["SINout"],
         label='ODE: 30 $\mathbf{Wm^{-2}}$',
         linewidth=5,
         color="#1d004f")

plt.plot(df_10["time"],
         df_10["SINout"],
         label='ODE: 8.76 $\mathbf{Wm^{-2}}$',
         linewidth=5,
         color="#ac86ef")

plt.plot(df_dyn["time"],
         df_dyn["SINout"],
         "--",
         label='Dynamic Irradiance',
         linewidth=5,
         color="#1d004f"
         )

plt.legend(fontsize=12)
plt.xlabel(r'$\mathbf{t}$ [days]', fontweight='bold', fontsize=20)
plt.ylabel(r'$\mathbf{S_{IN}}$ [g COD/m$^3$]', fontweight='bold', fontsize=20)
plt.xlim([0, 1])

fig3.savefig("graphs/SIN_evol.eps")
```




```
In [16]: # Graphing parameters
         from matplotlib.ticker import StrMethodFormatter


         set_color = "#444444"
         fig11 = plt.figure(figsize=(8,6), dpi=150)
         fig11.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.1f}')) # 1 decimal pla

         font = {'family' : 'DejaVu Sans',
                 'weight' : 'bold',
                 'size'   : 20}

         plt.rc('font', **font)

         # Graphing data
         plt.plot(data["time"][5:]/86400.0,
                  data["G850"][5:],
                  color=set_color,
                  label='FPR',
                  linewidth=5)
```

```
plt.plot(cstr["time"][2:]/86400.0,
         cstr["G850"][2:],
         ':',
         color=set_color,
         label='CR',
         linewidth=5)

plt.legend(fontsize=16)
plt.xlabel(r't [d]', fontweight='bold', fontsize=24)
plt.ylabel(r'$\mathbf{Irradiance}$ [Wm$^{-2}$]', fontweight='bold', fontsize=24)
plt.xlim([0, 1])
plt.tight_layout()
fig11.savefig("graphs/G850_evol.pdf")
```

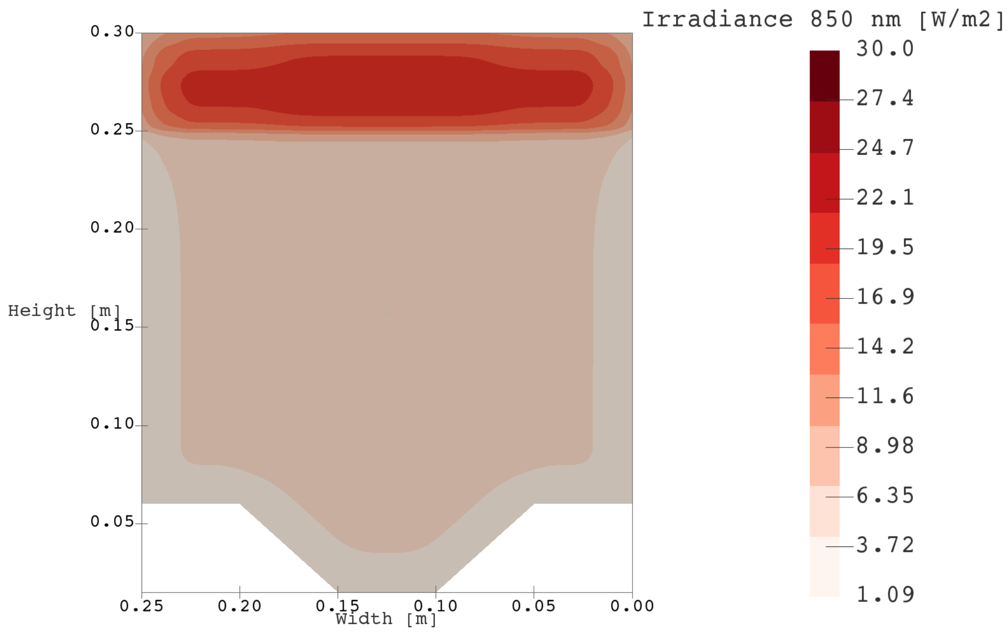



```
In [17]: print(f'cstr initial value is {cstr["G850"][2]} W/m2')
         print(f'flat plate reactor is {data["G850"][5]} W/m2')
```

```
cstr initial value is 5.0976597 W/m2
flat plate reactor is 11.2807 W/m2
```

```
In [18]: from IPython.core.display import Image
         Image(filename='./graphs/irradiance_distribution.png')
```

Out[18]:

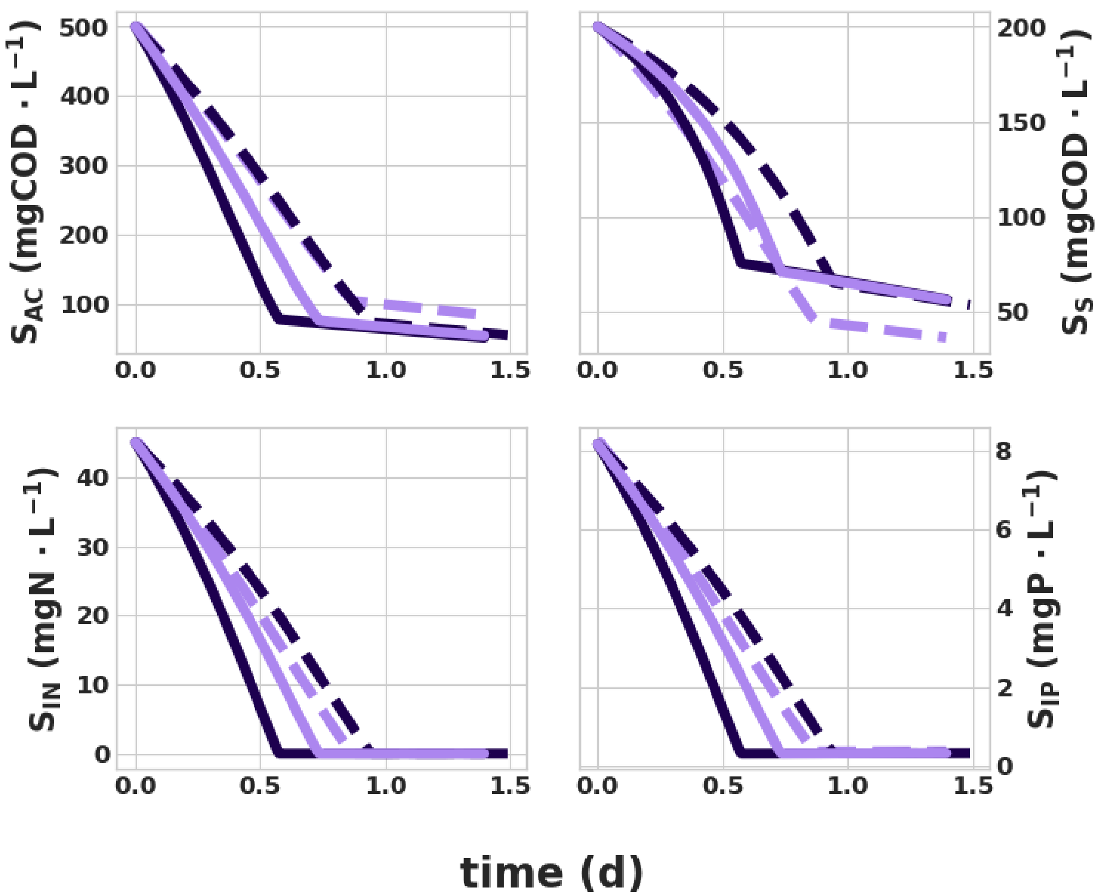



```
In [19]: import matplotlib as mpl
         mpl.rcParams['lines.linewidth'] = 5

         ## Plotting
         # plot for substrate concentration over time
         fig = plt.figure(figsize=(8,6), dpi=100)
         ax = fig.add_subplot(221)

         ax.tick_params(axis='both',
                        reset=False,
                        labelsize=12)

         plt.ylabel("$\mathbf{S_{AC}}$ ($\mathbf{mgCOD\, \cdot L^{-1}}$)",
                    fontsize=16, fontweight='bold')
         #plt.xlabel("time (d)", fontsize=20, fontweight='bold')


         plt.plot(data['time']/3600/24, data['SAC']*1000, '--',
                  label='CFD solution',
                  color='#ac86ef')

         plt.plot(df_dyn['time'], df_dyn['SACout'], '--',
```

```
        label='dynamic irradiance',
        color='#1d004f')

plt.plot(df_30['time'], df_30['SACout'],
        label='maximum nominal irradiance',
        color='#1d004f')

plt.plot(df_10['time'], df_10['SACout'],
        label='average irradiance',
        color='#ac86ef')

######################################################
######## SAC PLOT
######################################################
ax = fig.add_subplot(222)

ax.yaxis.tick_right()
ax.yaxis.set_label_position('right')

ax.tick_params(axis='both',
               reset=False,
               labelsize=12)

plt.ylabel("$\mathbf{S_{S}}$ ($\mathbf{mgCOD\, \cdot L^{-1}}$)",
          fontsize=16, fontweight='bold')

plt.plot(data['time']/3600/24, data['SS']*1000, '--',
        label='CFD solution',
        color='#ac86ef')

plt.plot(df_dyn['time'], df_dyn['SSout'], '--',
        label='dynamic irradiance',
        color='#1d004f')

plt.plot(df_30['time'], df_30['SSout'],
        label='maximum nominal irradiance',
        color='#1d004f')

plt.plot(df_10['time'], df_10['SSout'],
        label='average irradiance',
        color='#ac86ef')

######################################################
######## SIN PLOT
######################################################
ax = fig.add_subplot(223)

ax.tick_params(axis='both',
```

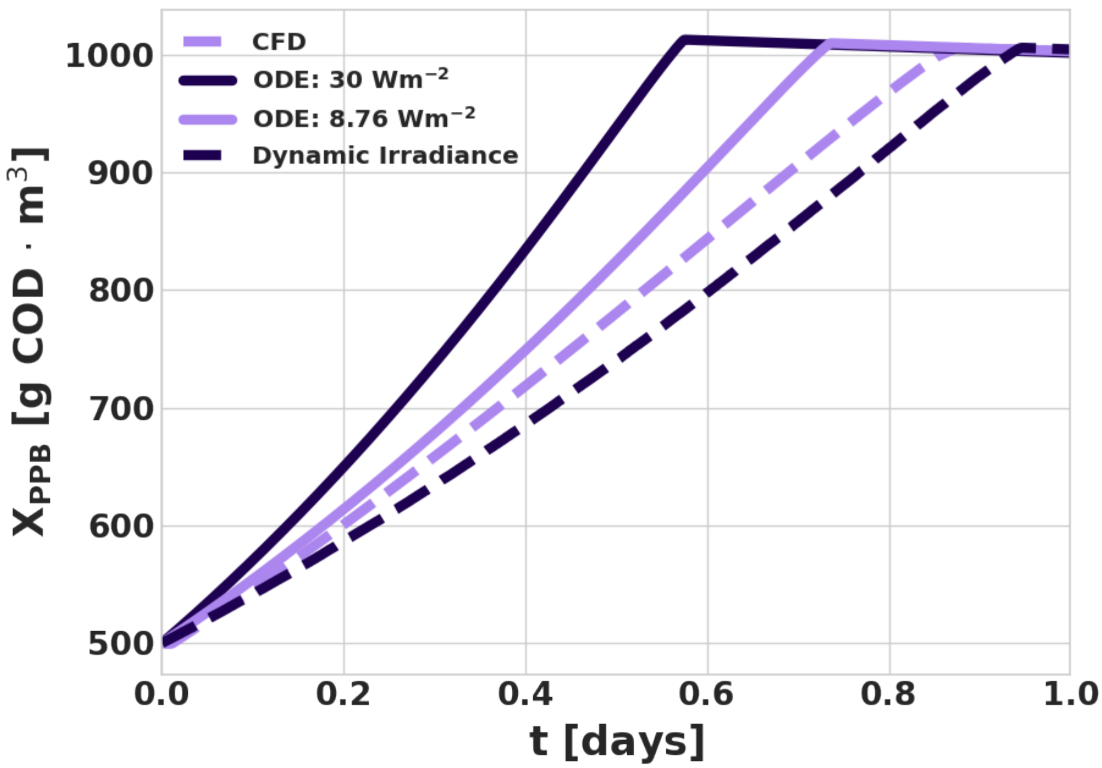
```
                reset=False,
                labelsize=12)
plt.ylabel("$\mathbf{S_{IN}}$ ($\mathbf{mgN\, \cdot L^{-1}}$)",
          fontsize=16, fontweight='bold')
#plt.xlabel("time (d)", fontsize=20, fontweight='bold')


plt.plot(data['time']/3600/24, data['SIN']*1000, '--',
        label='CFD solution',
        color='#ac86ef')

plt.plot(df_dyn['time'], df_dyn['SINout'], '--',
        label='dynamic irradiance',
        color='#1d004f')

plt.plot(df_30['time'], df_30['SINout'],
        label='maximum nominal irradiance',
        color='#1d004f')
4
plt.plot(df_10['time'], df_10['SINout'],
        label='average irradiance',
        color='#ac86ef')

######################################################
######## SIP PLOT
######################################################
ax = fig.add_subplot(224)

ax.yaxis.tick_right()
ax.yaxis.set_label_position('right')

ax.tick_params(axis='both',
                reset=False,
                labelsize=12)
plt.ylabel("$\mathbf{S_{IP}}$ ($\mathbf{mgP\, \cdot L^{-1}}$)",
          fontsize=16, fontweight='bold')
#plt.xlabel("time (d)", fontsize=20, fontweight='bold')


plt.plot(data['time']/3600/24, data['SIP']*1000, '--',
        label='CFD solution',
        color='#ac86ef')

plt.plot(df_dyn['time'], df_dyn['SIPout'], '--',
        label='dynamic irradiance',
        color='#1d004f')

plt.plot(df_30['time'], df_30['SIPout'],
```

```
        label='maximum nominal irradiance',
        color='#1d004f')

plt.plot(df_10['time'], df_10['SIPout'],
        label='average irradiance',
        color='#ac86ef')

fig.text(0.5, -0.05, 'time (d)', fontsize=20, fontweight='bold', ha='center')
#plt.xlabel("time (d)", fontsize=20, fontweight='bold')

fig.tight_layout()
fig.savefig('graphs/substrate_evolution.eps', bbox_inches='tight')
```




```
In [20]: fig3 = plt.figure(figsize=(8,6), dpi=120)
         ax1 = fig3.add_subplot(111)

         ax1.tick_params(axis="both",
                         reset=False,
                         labelsize=16)
```

```
font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 14}

plt.rc('font', **font)



plt.plot(data["time"]/86400,
         data["XPB"]*1000,
         "--",
         label='CFD',
         linewidth=5,
         color="#ac86ef"
         )


plt.plot(df_30["time"],
         df_30["XPBout"],
         label='ODE: 30 $\mathbf{Wm^{-2}}$',
         linewidth=5,
         color="#1d004f")

plt.plot(df_10["time"],
         df_10["XPBout"],
         label='ODE: 8.76 $\mathbf{Wm^{-2}}$',
         linewidth=5,
         color="#ac86ef")

plt.plot(df_dyn["time"],
         df_dyn["XPBout"],
         "--",
         label='Dynamic Irradiance',
         linewidth=5,
         color="#1d004f"
         )

plt.legend(fontsize=12)
plt.xlabel(r'$\mathbf{t}$ [days]', fontweight='bold', fontsize=20)
plt.ylabel(r'$\mathbf{X_{PPB}}$ [g COD $\cdot$ m$^3$]', fontweight='bold', fontsize=2(
plt.xlim([0, 1])

fig3.savefig("graphs/XPB_evol.eps")
```




```
In [21]: df_dyn.to_csv("./data/flat_dyn.csv")
         df_10.to_csv("./data/flat_10.csv")
         df_30.to_csv("./data/flat_30.csv")
         data.to_csv("./data/flat_cfd.csv")
```

```
In [22]: # Streamlines for the stirred vessel

         stream = pd.read_csv("./data/cylinder.csv")["GLambda_0"]
         streamtime = pd.read_csv("./data/cylinder.csv")["IntegrationTime"]
         print(f"Length of G850 is {len(stream)}.")
         print(f"Length of time array is {len(streamtime)}.")

         # This is for the flat plate reactor
         lst = []
         M = 500
         j = 0
         starter = 30
         j = starter

         while starter < len(streamtime)-2:
             while streamtime[j] < streamtime[j+1]:
                 j = j + 1
             if j - starter > M:
```

```
                lst.append([starter, j])
            starter = j + 1
            j = starter
```

```
Length of G850 is 223647.
Length of time array is 223647.
```

```
In [23]: # Determination of Law

         import scipy.stats as stats
         part0 = stream[lst[0][0]:lst[0][1]]
         time0 = streamtime[lst[0][0]:lst[0][1]]
         meanG = np.log(stream[lst[0][0]:lst[0][1]]).mean()
         stdG = np.log(part0).std()
         maxG = part0.max()
         minG = part0.min()

         particle_cyl = part0
         time_cyl = time0
         minG_cyl = minG


         loc_cyl, scale_cyl = stats.expon.fit(particle_cyl, floc=minG)
         x_cyl = np.linspace(0,30)
         dist_cyl = stats.expon.pdf(x_cyl, loc_cyl, scale_cyl)

         weights_cyl = np.ones_like(particle_cyl)/float(len(particle_cyl))


         #print(loc_cyl, scale_cyl)
         #print(f'rate = {1/scale_cyl}')
         #print(minG_cyl)
```

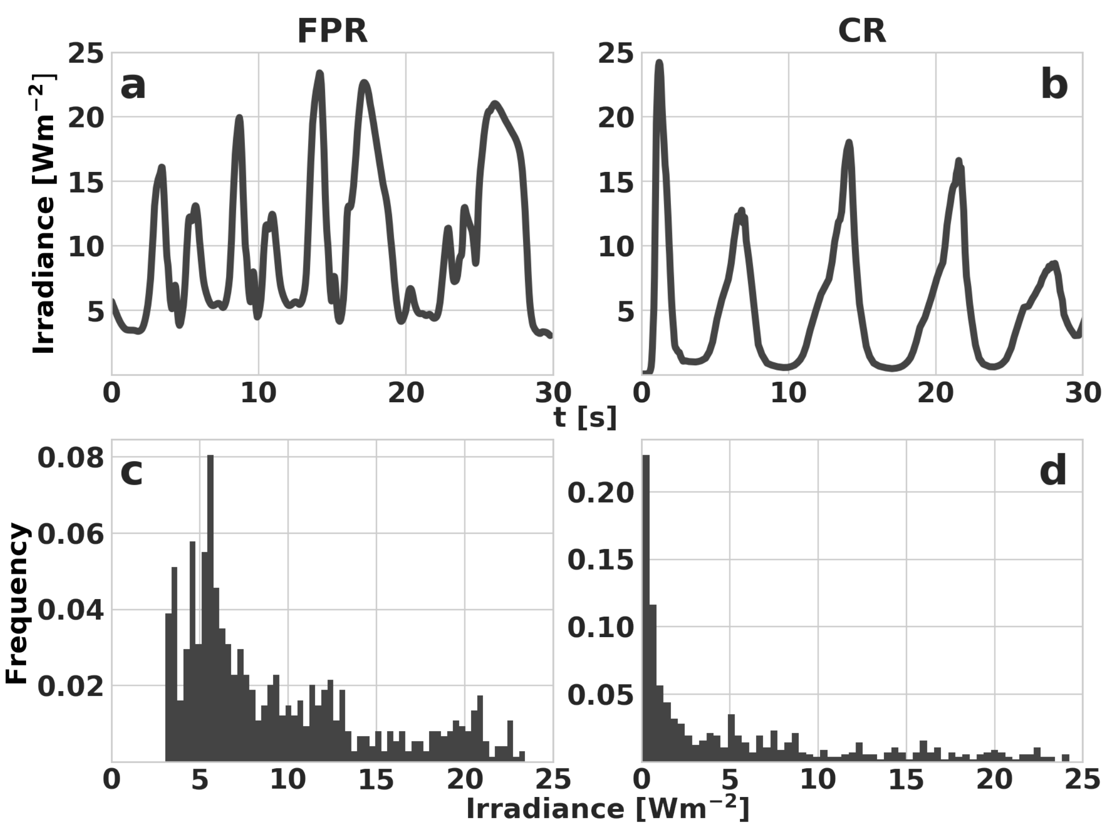
6 S.5 Summary of Short Term Dynamic Irradiance

The short term evolution of the radiative field, based on the flow field is summarised below. The first row corresponds to the time series evolution of the radiative field and the probability distribution (expon distribution) of an experienced irradiance for a particle for the flat plate reactor. The bottom row is the same, but for the cylinder.

```
In [24]: # Generate Graph for Publication
         import matplotlib as mpl
         mpl.rcParams['lines.linewidth'] = 5
         from matplotlib.ticker import MaxNLocator
         set_color = "#444444"
         plt.style.use("seaborn-whitegrid")
         font = {'family' : 'DejaVu Sans',
```

```
        'weight' : 'bold',
        'size'   : 16}

plt.rc('font', **font)

def format_ticks():
    plt.gca().yaxis.set_major_locator(MaxNLocator(prune='lower'))
#    plt.gca().xaxis.set_major_formatter(StrMethodFormatter('{x:,.1f}')) # 1 decimal p
    plt.locator_params(nbins=5)
    plt.locator_params(numticks=5)
#    if x == "xy":
#        plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.1f}')) # 1 deci

fig2 = plt.figure(figsize=(10,7.5), frameon=False, dpi=200)



ax = fig2.add_subplot(223)
ax.tick_params(axis='both',
              reset=False,
              labelsize=16)
plt.hist(particle_flat, 60, weights=weights_flat, color=set_color)
plt.ylabel('Frequency', fontsize=16, fontweight='bold', color="black")
plt.xlim([0, 25])
ax.text(20, -0.015, "$\mathbf{Irradiance\, \, [W m^{-2}]}$", fontsize=16)
format_ticks()


ax = fig2.add_subplot(224)
ax.tick_params(axis='both',
              reset=False,
              labelsize=16)
plt.hist(particle_cyl, 60, weights=weights_cyl, color=set_color)
plt.xlim([0, 25])
format_ticks()


ax = fig2.add_subplot(222)
ax.tick_params(axis='both',
              reset=False,
              labelsize=16)
plt.plot(streamtime[lst[0][0]:lst[0][1]], stream[lst[0][0]:lst[0][1]],
        color=set_color, linewidth=4)
plt.title("CR", fontweight='bold')
plt.xlim([0, 30])
plt.ylim([0, 25])
format_ticks()
```

```
ax = fig2.add_subplot(221)
ax.tick_params(axis='both',
              reset=False,
              labelsize=16)
plt.plot(streamtime_flat[lst_flat[1][0]:lst_flat[1][1]], stream_flat[lst_flat[1][0]:ls
        color=set_color, linewidth=4)
plt.title("FPR", fontweight='bold')
plt.xlim([0, 30])
plt.ylim([0, 25])
plt.ylabel('$\mathbf{Irradiance\, \, [W m^{-2}]}$', fontsize=16, color="black")
ax.text(30, -4.0, "t [s]", fontsize=16)
format_ticks()


ax.text(0.5,21.5,"a", fontsize=24, fontweight="bold")
ax.text(63,21.5,"b", fontsize=24, fontweight="bold")
ax.text(0.5,-8.5,"c", fontsize=24, fontweight="bold")
ax.text(63,-8.5,"d", fontsize=24, fontweight="bold")
fig2.savefig("graphs/G850_short.pdf")
plt.show()
```



In [25]:
```
time0.index = pd.RangeIndex(len(time0.index))
part1 = np.concatenate(60*24*3*[part0])
time1 = np.concatenate(60*24*3*[time0])

# Create a deltaTime list
deltaTime = []
for i in range(len(time0)-1):
    deltaTime.append(time0[i+1] - time0[i])

time1 = [0]

# Turn deltaTime into a np.array
deltaTime = np.concatenate((60*24*3+4)*[np.array(deltaTime)])


for i in range(len(deltaTime)):
    time1.append(time1[-1]+deltaTime[i])

#for i in range(3*60*24):
len(deltaTime)

time2 = np.asarray(time1)/86400
f = interp1d(time2, part1[0:len(time1)])
Nsteps = 800
```

In [26]:
```
#########################################################
# USER INPUT
num_steps = 10000

days = 172800/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45  # 45
SIP0 = 8.15   # 8.15
SI0 = 150.0
XPB0 = 500.0
XS0 = 300.0
XI0 = 100.0

##########################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

# Interpolate the irradiance array

t_new = np.linspace(0, time2.max(), num_steps)
G850 = f(t_new)
```

```
# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps-1):
    t = [t_new[step], t_new[step+1]]

    # Get G850 for this step
    u = G850[step:step+1]

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(u,))

    y_inter = np.float64(soln[1])

    y_out.append(y_inter)

    y0 = y_inter.tolist()

y_out = np.asarray(y_out)
cyl_dyn = pd.DataFrame((y_out))

cyl_dyn['SSout'] =  y_out[:,0]
cyl_dyn['SACout'] = y_out[:,1]
cyl_dyn['SICout'] = y_out[:,2]
cyl_dyn['SH2out'] = y_out[:,3]
cyl_dyn['SINout'] = y_out[:,4]
cyl_dyn['SIPout'] = y_out[:,5]
cyl_dyn['SIout'] = y_out[:,6]
cyl_dyn['XPBout'] = y_out[:,7]
cyl_dyn['XSout'] = y_out[:,8]
cyl_dyn['XIout'] = y_out[:,9]
cyl_dyn['time'] = t_new
```

In [27]:
```
##########################################################
# USER INPUT
num_steps = 800

days = 120604/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45  # 45
SIP0 = 8.15   # 8.15
SI0 = 150.0
```

```
XPB0 = 500.0
XS0 = 300.0
XI0 = 100.0

##########################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

time_ode = np.linspace(0,days,num_steps)
#u_t = interp1d(time_input.squeeze(), full_G850.squeeze())

# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps - 1):
    t = [time_ode[step], time_ode[step+1]]

    # Get G850 for this step
    G850 = 30

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(G850,))

    y_inter = np.float64(soln[1])

    y_out.append(y_inter)

    y0 = y_inter.tolist()


y_out = np.asarray(y_out)
cyl_30 = pd.DataFrame(y_out)
cyl_30['SSout'] =  y_out[:,0]
cyl_30['SACout'] = y_out[:,1]
cyl_30['SICout'] = y_out[:,2]
cyl_30['SH2out'] = y_out[:,3]
cyl_30['SINout'] = y_out[:,4]
cyl_30['SIPout'] = y_out[:,5]
cyl_30['SIout'] = y_out[:,6]
cyl_30['XPBout'] = y_out[:,7]
cyl_30['XSout'] = y_out[:,8]
cyl_30['XIout'] = y_out[:,9]
cyl_30['time'] = time_ode
```

In [28]:
```
#########################################################
# USER INPUT
num_steps = 800
```

```python
days = 120604/3600/24
SS0 = 200.0
SAC0 = 500.0
SIC0 = 7.0E-6
SH20 = 0.0
SIN0 = 45  # 45
SIP0 = 8.15   # 8.15
SI0 = 150.0
XPB0 = 500.0
XS0 = 300.0
XI0 = 100.0

##########################################################
y0 = [SS0, SAC0, SIC0, SH20, SIN0, SIP0, SI0, XPB0, XS0, XI0,]

time_ode = np.linspace(0,days,num_steps)
#u_t = interp1d(time_input.squeeze(), full_G850.squeeze())

# Store our state outputs here
y_out = [y0]

# Loop over each point in the dataset

for step in range(num_steps - 1):
    t = [time_ode[step], time_ode[step+1]]

    # Get G850 for this step
    G850 = beer_lambert(30, 53+15, 0.015)

    # Solve the ODEs
    soln = odeint(pam_batch, y0, t, args=(G850,))

    y_inter = np.float64(soln[1])

    y_out.append(y_inter)

    y0 = y_inter.tolist()


y_out = np.asarray(y_out)
cyl_10 = pd.DataFrame(y_out)
cyl_10['SSout'] =  y_out[:,0]
cyl_10['SACout'] = y_out[:,1]
cyl_10['SICout'] = y_out[:,2]
cyl_10['SH2out'] = y_out[:,3]
cyl_10['SINout'] = y_out[:,4]
cyl_10['SIPout'] = y_out[:,5]
```

```
cyl_10['SIout'] = y_out[:,6]
cyl_10['XPBout'] = y_out[:,7]
cyl_10['XSout'] = y_out[:,8]
cyl_10['XIout'] = y_out[:,9]
cyl_10['time'] = time_ode
```

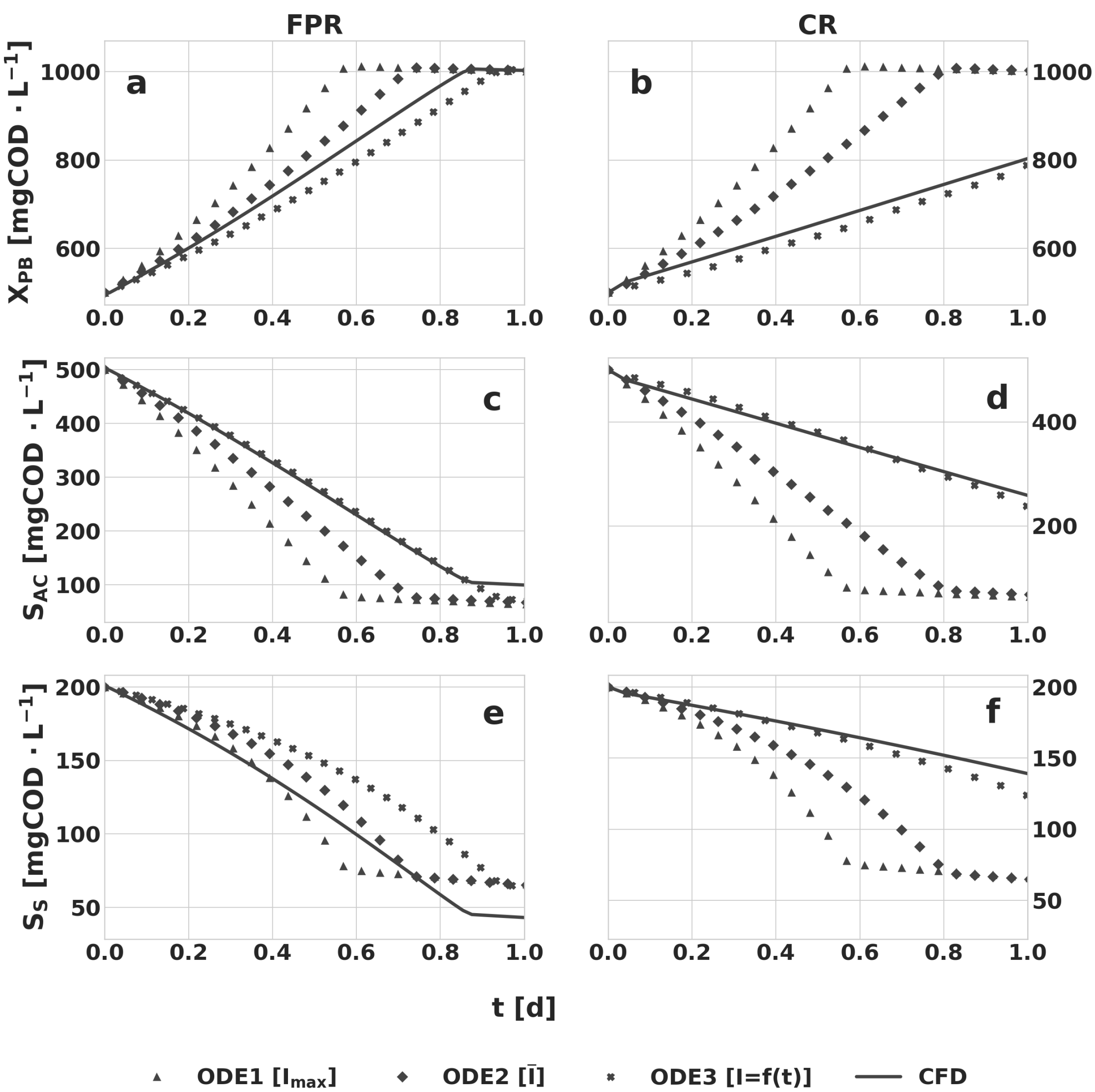
In [29]:
```
# Import CFD Solution of the stirred vessel.
cyl_cfd = pd.read_csv("./data/round_cfd.csv")
cyl_cfd.columns = ["index", "time",
                   "SS", "SAC", "SIN", "SIP",
                   "XPB"]
```

In [30]:
```
##################################################################
######      PLOT XPB FOR CR ####################################

plt.rcParams['lines.linewidth'] = 3.0
dark = "#444444"

# Set the plot styles here
def CFDplot(x, y, shade):
    plt.plot(x, y, label="CFD", color=shade)

def ODE1plot(x, y, shade):
    style = "^"
    plt.plot(x.iloc[::25], y.iloc[::25], style,
             label="$\mathregular{ODE1\, \, [I_{max}]}$",
             color=shade)

def ODE2plot(x, y, shade):
    style = "D"
    plt.plot(x.iloc[::25], y.iloc[::25], style,
             label="$\mathregular{ODE2\, \, [\overline{I}]}$",
             color=shade)

def ODE3plot(x, y, shade):
    style = "X"
    plt.plot(x.iloc[::250], y.iloc[::250], style,
             label="ODE3 [I=f(t)]",
             color=shade)


fig1 = plt.figure(figsize=(15, 15), dpi=180)
fig1.set_edgecolor("black")

ax1 = fig1.add_subplot(321)

ax1.tick_params(axis='both',
               reset=False,
```

```
                labelsize=20)
plt.ylabel("$\mathbf{X_{PB}}$ [$\mathbf{mg COD\, \cdot L^{-1}}$]",
          fontsize=24, fontweight='bold')


ODE1plot(df_30['time'], df_30['XPBout'], dark)
ODE2plot(df_10['time'], df_10['XPBout'], dark)
ODE3plot(df_dyn['time'], df_dyn['XPBout'], dark)
CFDplot(data['time']/3600/24, data['XPB']*1000, dark)

plt.title("FPR", fontsize=24, fontweight='bold')
plt.xlim([0,1])
plt.text(0.05, 950, "a", fontsize=30)

###################################################################
######      PLOT XPB FOR CR ####################################

ax2 = fig1.add_subplot(322, sharex=ax1, sharey=ax1)
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position('right')

ODE1plot(cyl_30['time'], cyl_30['XPBout'], dark)
ODE2plot(cyl_10['time'], cyl_10['XPBout'], dark)
ODE3plot(cyl_dyn['time'], cyl_dyn['XPBout'], dark)
CFDplot(cyl_cfd['time']/3600/24, cyl_cfd['XPB']*1000, dark)
plt.xlim([0,1])
ax2.tick_params(axis='both',
               reset=False,
               labelsize=20)
plt.title("CR", fontsize=24, fontweight='bold')

plt.text(0.05, 950, "b", fontsize=30)
###################################################################
######      PLOT SAC FOR FPR ####################################
ax3 = fig1.add_subplot(323)


ax3.tick_params(axis='both',
               reset=False,
               labelsize=20)

plt.ylabel("$\mathbf{S_{AC}}$ [$\mathbf{mg COD\, \cdot L^{-1}}$]",
          fontsize=24, fontweight='bold')


ODE1plot(df_30['time'], df_30['SACout'], dark)
ODE2plot(df_10['time'], df_10['SACout'], dark)
ODE3plot(df_dyn['time'], df_dyn['SACout'], dark)
```

```
CFDplot(data['time']/3600/24, data['SAC']*1000, dark)

plt.xlim([0,1])
plt.text(0.9, 425, "c", fontsize=30)
#########################################################3
######## PLOT SAC FOR CR
#########################################################
ax4 = fig1.add_subplot(324, sharex=ax2)
ax4.yaxis.tick_right()
ax4.yaxis.set_label_position('right')

ODE1plot(cyl_30['time'], cyl_30['SACout'], dark)
ODE2plot(cyl_10['time'], cyl_10['SACout'], dark)
ODE3plot(cyl_dyn['time'], cyl_dyn['SACout'], dark)
CFDplot(cyl_cfd['time']/3600/24, cyl_cfd['SAC']*1000, dark)
plt.xlim([0,1])
ax4.tick_params(axis='both',
               reset=False,
               labelsize=20)


plt.text(0.9, 425, "d", fontsize=30)

#################################################################
######### PLOT SS FOR FPR
################################################################
ax5 = fig1.add_subplot(325)

ax5.tick_params(axis='both',
               reset=False,
               labelsize=20)

plt.ylabel("$\mathbf{S_{S}}$ [$\mathbf{mgCOD\, \cdot L^{-1}}$]",
          fontsize=24, fontweight='bold')


ODE1plot(df_30['time'], df_30['SSout'], dark)
ODE2plot(df_10['time'], df_10['SSout'], dark)
ODE3plot(df_dyn['time'], df_dyn['SSout'], dark)
CFDplot(data['time']/3600/24, data['SS']*1000, dark)
plt.xlim([0, 1])
plt.text(0.9, 175, "e", fontsize=30)

#################################################################
######### PLOT SS FOR CF
################################################################
ax6 = fig1.add_subplot(326)
ax6.yaxis.tick_right()
```

```
ax6.yaxis.set_label_position('right')
ax6.tick_params(axis='both',
                reset=False,
                labelsize=20)


ODE1plot(cyl_30['time'], cyl_30['SSout'], dark)
ODE2plot(cyl_10['time'], cyl_10['SSout'], dark)
ODE3plot(cyl_dyn['time'], cyl_dyn['SSout'], dark)
CFDplot(cyl_cfd['time']/3600/24, cyl_cfd['SS']*1000, dark)
plt.xlim([0, 1])
plt.text(0.9, 175, "f", fontsize=30)
plt.legend(loc=(-1.15, -0.6), ncol=4, fontsize=20)
fig1.text(0.45, 0.060, "t [d]",  fontsize=24)
fig1.savefig("graphs/growth_kinetics.pdf")
```



7 S.6 Long-term dynamic irradiance

One of the final mechanisms to discuss is the long term evolution of the irradiance field as the biomass grows. The figures have been exported to the paper document, and also pasted below. Note that there have only been 10 levels of discretisation for the colour scheme. It has been anecdotally found that having too fine a resolution when displaying color graphs can detract from the physics of the system.

```
In [31]: from IPython.core.display import Image
         Image(filename='graphs/G850_evol_long.png')
```

Out[31]: