	TOC	$\delta^{13}C$	C/N
Curve equation form	**aln(bx+c)+d**	**aln(bx+c)+d**	**aln(bx)+c**
a	**0.182**	**-0.208**	**0.167**
b	**2.267**	**0.931**	**25.612**
c	**0.01217**	**0.01303**	**-0.022**
d	**0.781**	**0.128**	**/**
r^2	**0.909**	**0.63**	**0.59**
Number of values for partial curve fitting	**13**	**17**	**11**
Limit depth (cm)	**125**	**155**	**105**
Limit age (ka cal BP)	**1.65**	**2.5**	**1.35**

Table S1 : Curve fit parameters

The following figure presents the "raw" data depending on depth (cm).

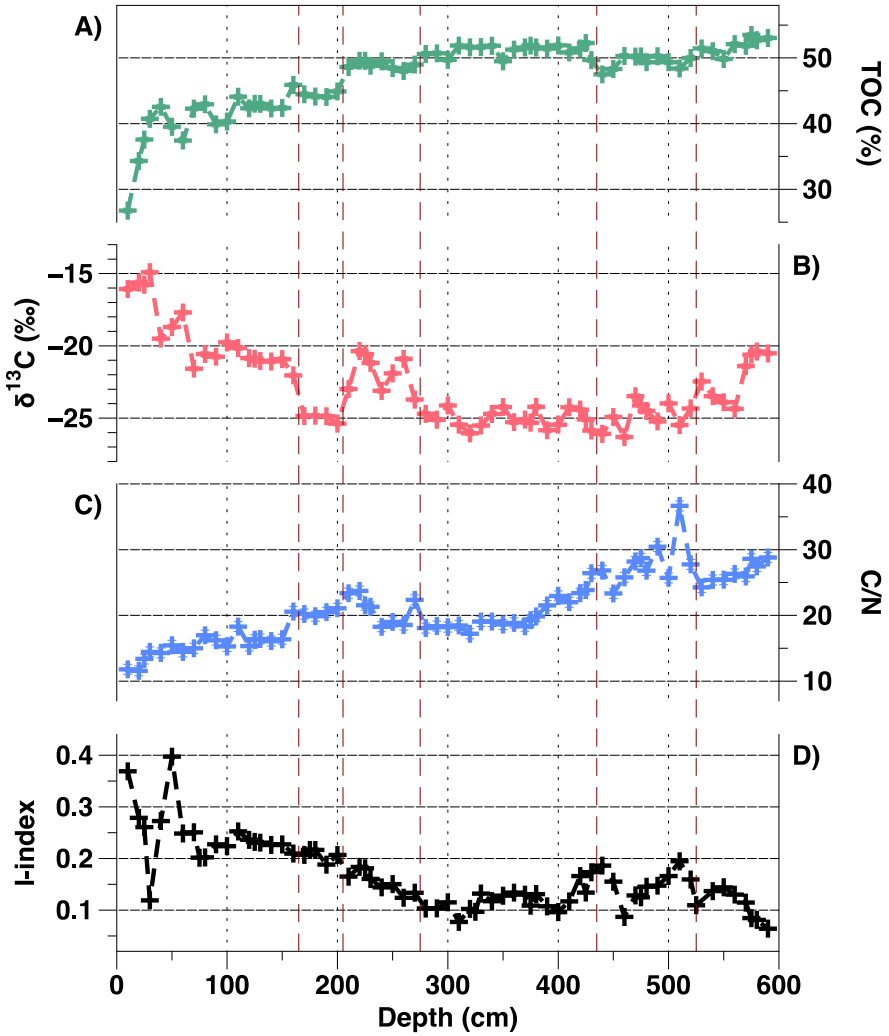

Figure S2 : Raw data (ie before data processing) from the NGaoundaba peat deposit: (A) TOC (%), (B) δ13Cbulk (h), (C) C/N from Elemental and stable isotope analysis and the I-index from Rock-Eval® analysis depending on depth (cm). Visible transitions are indicated with vertical red dashed lines.

The following pages present the code used for the modeling in this paper for TOC values, the same code is used for d13C and C/N after replacing the corresponding terms. The code needs to be adapted depending on the chosen form of the function. The moving average method was used in the paper but three methods were tested and are presented here.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import csv
from scipy.optimize import curve_fit

from sklearn.preprocessing import MinMaxScaler #For data normalization
import statsmodels.api as sm #For data smoothing (Lowess)
from scipy.signal import savgol_filter #For data smoothing (Sawitzky-Golay)

## Objective function
def function(x,a,b,c,d):
    return a*np.log(b*x+c)+d
def function2(x,a,b,c):
    return a*np.log(b*x)+c

## Entering the file and dataplot
fig = plt.figure()

print('Attention ! Your columns has to be named as age, depth, %C,...')
filename=input('What is your file name (/!\Dont forget the.csv) ?')

data=pd.read_csv(filename, delimiter=";")

data.plot.scatter(x='age', y='%C', marker='+') # for scatter point

print('Please, save and/or close figure files to continue')
plt.legend()
plt.show()

## Normalization (between 0 and 1) of the data
# Normalization of y-data
minmaxscale = MinMaxScaler().fit(data['%C'].values.reshape(-1,1))
data['%Cstd']=minmaxscale.transform(data['%C'].values.reshape(-1,1))
# Normalization of x-data
minmaxscale = MinMaxScaler().fit(data['age'].values.reshape(-1,1))
data['agestd']=minmaxscale.transform(data['age'].values.reshape(-1,1))
data['agestd']=data['agestd']+0.001

## Curve fit - normalized data
fig2 = plt.figure()
```

```
data.plot.scatter(x='agestd', y='%Cstd', marker='+') # for scatter point

Column_agestd = data.columns.get_loc("agestd") #Finding the right columns for the age
Column_percentC = data.columns.get_loc("%Cstd") #Finding the right columns for the
percentC

datav=data.to_numpy()

#Based on whole dataset
xdata0=datav[:,Column_agestd]
ydata0=datav[:,Column_percentC]
xdata0=np.array(xdata0, dtype=[('O', float)]).astype(float) # in case the values are not
considered as float by default
ydata0=np.array(ydata0, dtype=[('O', float)]).astype(float)

# Estimating optimizing parameters #if needed if encountering the error "Optimal
parameters not found: Number of calls to function has reached maxfev = 800" or possibility
to change the "maxfev" parameter
a0=ydata0[0]
amax=max(ydata0)
th=xdata0[np.searchsorted(-ydata0+a0, -0.5*amax)]
b0 = np.log(2)/th

popt, _ = curve_fit(function, xdata0, ydata0, p0=(a0,b0,0,0))
#popt, _ = curve_fit(function, xdata0, ydata0)
#popt, _ = curve_fit(function2, xdata0, ydata0)
a, b, c ,d= popt
#a, b, c = popt

print('y=%.5f*np.log(%.5f*x+%.5f)+%.5f'%(a,b,c,d))
#print('y=%.5f*np.log(%.5f*x)+%.5f'%(a,b,c))

residuals0 = ydata0- function(xdata0, *popt) # for r2 calculation
#residuals0 = ydata0- function2(xdata0, *popt) # for r2 calculation

ss_res0 = np.sum(residuals0**2)
ss_tot0 = np.sum((ydata0-np.mean(ydata0))**2)
r_squared0 = 1 - (ss_res0 / ss_tot0)
print(r_squared0)

poptr2 = (a, b, c , d, r_squared0) # list for the legend
#poptr2 = (a, b, c , r_squared0) # list for the legend

plt.plot(xdata0, function(xdata0, *popt), 'r--',label='f(x)=%5.3f*ln(%5.3f*x+%.5f) + %5.3f,
r\u00B2=%5.3f' % tuple(poptr2), linewidth=3)
#plt.plot(xdata0, function2(xdata0, *popt), 'r--',label='f(x)=%5.3f*ln(%5.3f*x) + %5.3f,
r\u00B2=%5.3f' % tuple(poptr2), linewidth=3)
```

```
#Based on data with f'(x)>1
max_age = abs((a*b-c)/b) # Finding the age corresponding to f'(x)>1 – function 1
#max_age = abs(a) # Finding the age corresponding to f'(x)>1 – function 2

line_max_age = 0
while xdata0[line_max_age] <= max_age : #Finding the corresponding line
   line_max_age = line_max_age + 1

xdata1=datav[0:line_max_age,Column_agestd] #Selecting only these lines in the dataset
ydata1=datav[0:line_max_age,Column_percentC]

# Estimating optimizing parameters #if needed if encountering the error "Optimal
parameters not found: Number of calls to function has reached maxfev = 800" or possibility
to change the "maxfev" parameter
b0=0.5**((ydata0[1]-ydata0[0])/(ydata0[1]-ydata0[2])) #Using the 3 first lines to estimate a,
b and c, considering d=0
a0=(ydata0[1]-ydata0[2])/np.log(1/2)
#c0=ydata0[0]/a0
#or c0=ydata0[0] #depending on what is working the best

popt, _ = curve_fit(function2, xdata1, ydata1, p0=(a0,b0,0))
#popt, _ = curve_fit(function, xdata1, ydata1)
#popt, _ = curve_fit(function2, xdata1, ydata1)

a, b, c, d= popt
#a, b, c= popt

print('y=%.5f*np.log(%.5f*x+%.5f)+%.5f'%(a,b,c,d))
#print('y=%.5f*np.log(%.5f*x)+%.5f'%(a,b,c))

residuals1 = ydata1- function(xdata1, *popt) # for r2 calculation
#residuals1 = ydata1- function2(xdata1, *popt) # for r2 calculation
ss_res1 = np.sum(residuals1**2)
ss_tot1 = np.sum((ydata1-np.mean(ydata1))**2)
r_squared1 = 1 - (ss_res1 / ss_tot1)
print(r_squared1)

residuals1 = ydata0- function(xdata0, *popt) # for r2 calculation
#residuals1 = ydata0- function2(xdata0, *popt) # for r2 calculation
ss_res1 = np.sum(residuals1**2)
ss_tot1 = np.sum((ydata0-np.mean(ydata0))**2)
r_squared1 = 1 - (ss_res1 / ss_tot1)
print(r_squared1)
```

```
poptr2 = (a, b, c, d, r_squared1) # list for the legend
#poptr2 = (a, b, c, r_squared1) # list for the legend

plt.plot(xdata0, function(xdata0, *popt), 'g-.',label='f(x)=%5.3f*ln(%5.3f*x+%.5f) +%5.3f,
r\u00B2=%5.3f' % tuple(poptr2), linewidth=3)
#plt.plot(xdata0, function2(xdata0, *popt), 'g-.',label='f(x)=%5.3f*ln(%5.3f*x) +%5.3f,
r\u00B2=%5.3f' % tuple(poptr2), linewidth=3)

print('Please, save and/or close figure files to continue')

plt.legend()
plt.show()

## Correction

#data['corr%Cstd']=data['%Cstd']-a*np.log(b*data['agestd']+c)-d
data['corr%Cstd']=data['%Cstd']-a*np.log(b*data['agestd'])-c

fig3 = plt.figure()
#data.plot.line(x='agestd', y='corr%Cstd', label='data', style='--+', linewidth=3) # for scatter
point with dashed line
data.plot.line(x='agestd', y='corr%Cstd', style='--+') # for scatter point with dashed line

#data.plot.scatter(x='agestd', y='corr%Cstd', label='Normalized data', marker='+', s=40) # for
scatter point
 # for scatter point // data.plot(x='agestd', y='corr%Cstd') # for solid line
print(data['corr%Cstd'])
print('Please, save and/or close figure files to continue')
plt.show()

## Datasmoothing
fig4 = plt.figure()
data.plot.line(x='age', y='corr%Cstd', label='data', style='+') # for scatter point with dashed
line

# LOWESS method
lowess = sm.nonparametric.lowess(data['corr%Cstd'], data['age'], frac=0.1)
data['lowess']=lowess[:,1]
plt.plot(lowess[:, 0], lowess[:, 1], 'g-', label='Lowess method')

# Moving average box by convolution
datav=data.to_numpy()
Column_percentCcorr = data.columns.get_loc("corr%Cstd") #Finding the right columns for
the percentC

def smooth(y, box_pts): #The higher box_pts is, the smoother the signal is
    box = np.ones(box_pts)/box_pts
```

```
    y_smooth = np.convolve(y, box, mode='same')
    return y_smooth

plt.plot(datav[:,Column_age], smooth(datav[:,Column_percentCcorr],2), 'b-', label='Moving
average box by convolution')
plt.plot(datav[:,Column_age], smooth(datav[:,Column_percentCcorr],3), 'r-', label='Moving
average box by convolution')

data['movingaverage2']=smooth(datav[:,Column_percentCcorr],2)
data['movingaverage3']=smooth(datav[:,Column_percentCcorr],3)

# Savitzky-Golay method
window_size = 11
ysavit = savgol_filter(data['corr%Cstd'], window_size, 5) # window size must be odd,
polynomial order 5
plt.plot(datav[:,Column_age], ysavit, label='Savitzky-Golay method')
data['SGw11poly5']=ysavit

filename=input('How do you want to name your file (Don\'t forger the .csv) ')
data.to_csv(filename, sep=';')

plt.legend()
plt.title('Smoothing')
plt.show()
```