Supplementary Information: High Throughput Machine Learning-based Method to Charachterize Vitrification Kinetics

Anabella A. Abate,[†] Daniele Cangialosi,[*,‡] and Simone Napolitano[*,†]

†*Laboratory of Polymer and Soft Matter Dynamics, Experimental Soft Matter and Thermal Physics (EST), Université libre de Bruxelles (ULB), Bruxelles 1050, Belgium*

‡*Centro de Física de Materiales (CSIC-UPV/EHU), Paseo Manuel de Lardizabal 5, 20018 San Sebastián, Spain*

E-mail: daniele.cangialosi@ehu.eus; snapolit@ulb.ac.be

This Supplementary Material includes the following sections:

A) VITRIFAST: The Python Code of the Method to analyze the FSC data

B) The Python Code of the Method to obtain T_{ON} from FSC Measurements

C) The Phyton Code used to include White Noise to the FSC Measurements

D) The parameters employed to analyze the examples showed in the main text

In order to run the Python codes listed below first it's necessary to download and install the free software Anaconda:

https://www.anaconda.com/products/individual

Then it is required to launch the program Jupyter Notebook and select the folder where it's located the Phyton Code and the Excel file to analyze. The Excel file to analyze should be arrange in a specific way: the first two columns correspond to the reference sample; the first column corresponds to the temperature and the second one to the Heat Flow. Then, the rest of the columns correspond to the different aged experiments to be analyzed alternating between temperature and Heat Flow. It's important to add the information of each aged column as a header, for example the amount of aged time or the cooling velocity in order to later identify in the output the corresponding data.

There are two important parts in the Code: one it's the main program, described in the following section: "A) VITRIFAST: The Python Code of the Method to analyze the FSC data". For running that code it's recommended to first run the second Method showed in the second section "B) The Python Code of the Method to obtain T_{ON} from FSC Measurements". The method on Section B) will determine the value of T_{ON} using only the most aged sample of the set to analyze. Later, the program VITRIFAST will use that value as an input for the analysis of the whole experiment set. It's important to note that also the value of T_{ON} could be selected by hand by the user without using the code presented in section B).

We also submit the latest version of the code and a few examples in the following GitHub link:

https://github.com/anabate/VITRIFAST

A) VITRIFAST: The Python Code to analyze the FSC data

Fig. 1 shows all the parameters that are needed as an input to run the VITRIFAST program. The input parameters are:

L: the value of the row (the lowest index) to consider the data to analyze.

H: the value of the row (the higher index) to consider the data to analyze.

T_{min}: The value of the temperature T_{ON} obtained from the method in section B) or set by the user. The program will use this value as a lower limit for the integration in the matching areas to determine T_f, since the integration starts from T_{min}.

T_{max}: the value of the temperature that determines the end of the glass transition. The program will use this value in the integration of the matching areas to determine T_f, since the integration ends in T_{max}.

T_{glass}: The value of the temperature where we could say *by eye* that ends the glass line.

T_{melt}: The value of the temperature where we could say *by eye* that the liquid line starts.

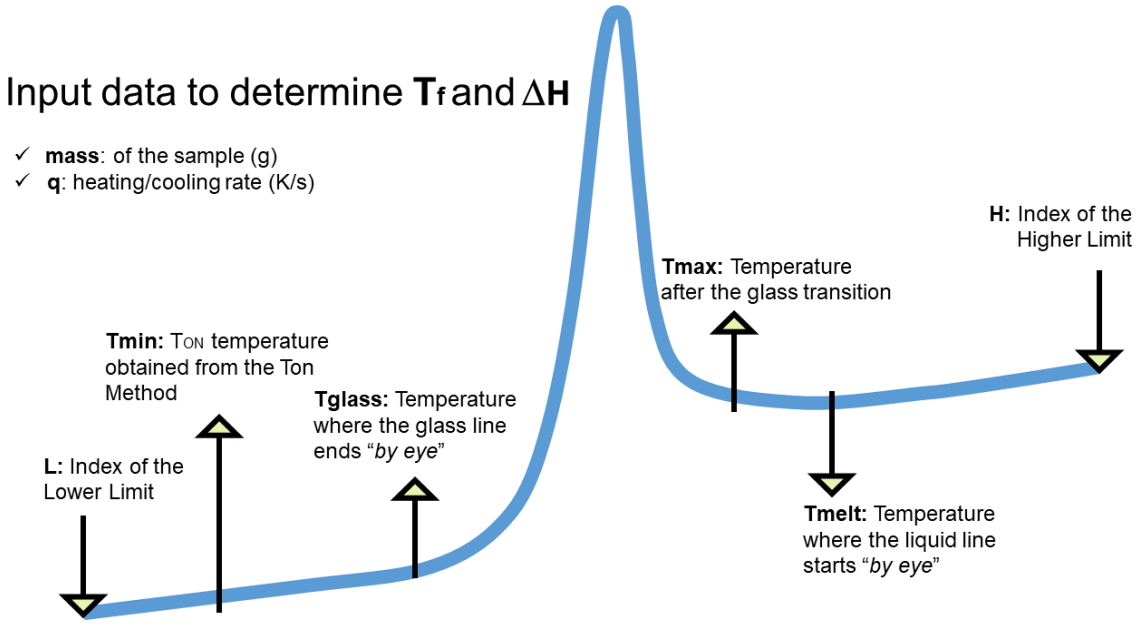


Figure 1: Input parameters needed in order to run VITRIFAST and determine T_f and ΔH.

The outputs of the program are two Excel files and one graph with all the corrected curves of c_p^{aged} as a function of T, including also the glass and the liquid lines employed for the calculations.

One Excel file contains the temperature and corrected c_p and Δc_p. The other Excel file contains the calculated parameters, which are:

c (a): Value obtained from the vertical shift of the aged c_p^{aged} data to match with the reference c_p^{ref}.

d (b): Value obtained from the slope change of the aged c_p^{aged} data to match with the reference c_p^{ref}.

MSE: The mean squared error obtained from the correction of the aged c_p^{aged} data with the reference c_p^{ref} using those values of c(a) and d(b).

T_f: the obtained value of T_f (in Kelvin) after performing the matching area method taking into account the areas of the corrected aged c_p^{aged} curves in the temperature range between T_{min} and T_{max}.

ΔH: the obtained value of the excess of enthalpy, ΔH, (in J/g) calculated as the area of the difference between the aged c_p^{aged} curve corrected and the reference c_p^{ref}.

Running time: the total amount of time (in seconds) that the program took to do all the analysis for each aged sample.

```
#MAIN CODE TO ANALYZE THE FSC DATA
#loading the librares needed
import pandas as pd
from scipy.optimize import least_squares
import numpy as np
from sklearn.linear_model import LinearRegression
from scipy.integrate import simps
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.pyplot import figure

#What we need is an Excel Spreadsheet where the first two columns are the Temperature and the
#Cp of the Reference and the remaining columns are Temperature and the Cp of the age samples.
#The Input file is T_Ref/Cp_Ref/T_Aged/Cp_Aged/.../T_Aged/Cp_Aged
```

```
#The first line is the Header and this program will save that name in the Output,
#so it's important to define it
filename = 'Name_of_the_file_to_analyze.xlsx'
#necessary based on the sheet called
sheet = 0

#Loading the data from the Input
data = pd.read_excel(filename, sheet_name=sheet, header=0)

#INPUT parameters_____*******Those values are part of an example********
#Select the Lower and  Higher Row Limit of the Experiment
#These should be the same for one set of aging experiments (reference and aged samples)
L = 20   #Index Lower limit
H = 1500  #Index higher limit
#Select the Limits in Temperature (in K) where the glass transition occurs.
Tmin = 280   #Tmin: Lower Limit: here you can use the TON obtained from the other method.
#If not, select a value  far below the transition
Tmax = 445   #Tmax: Upper Limit after the glass transition
Tglass = 310 #Tglass: Is the maximal T where ends the glass line by eye
Tmelt = 440  #Tmelt: Is the minimal T where starts the liquid line by eye
#********Those values are part of an example*******

#Initial seeds for the values of c (or a) and d (or b)-> initially both in zero
c = 0.0
d = 0.0
theta0 = [c, d]

# Slice the data within the Selected Range
data = data[L:H+1].reset_index()

#Number of columns to analyze (start to count from 1):
#Number of aged experiments to analyze  (multiple of 2 always)
Ncols = XX

#Conversion from Celsius to Kelvin (if needed), else set  tempOffset to zero.
tempOffset = 273.15
tempCols = [x for x in range(1, Ncols+1, 2)]
data.iloc[:, tempCols]  = data.iloc[:, tempCols] + tempOffset

# Creation of the variables we want to save
CHI = [] #Error
C = []   #c
D = []   #d
TF = []  #Fictive Temperature
I = []   #Name of each column
```

```
DH = []  #Recovery Entalpy
RUNNINGTime = []  #Time

#CONVERTION OF THE HF DATA TO CP (if needed)
#For FSC Measurements: conversion from HF (IN MILIWATTS) to Cp
#(HF divided by the cooling rate and the mass)
mass = 1.6e-7 # # in g
q = 1000 # in K/s
CpCols = [x for x in range(2, Ncols, 2)] # HF in mW
data.iloc[:, CpCols]  = data.iloc[:, CpCols] / (1000 * mass * q)
# divided by 1000 to convert to J/g.K


# Model used
def modelPoint(o, c, d, Tend, Tini, data, m, n, a, b):
    cp    = data.iloc[o, m]  #cp(T)

    DeltaT = Tend - Tini #Delta_T
    coef   = (1.0 - (d / a)) / DeltaT  #  (1- d/Cp(T=Tini)) / Delta_T
    e      = ( coef *  cp + b - Tend * coef ) / b;
    # ( coef * cp(T) + Cp(T=Tend) -Tend *coef ) / Cp(T=Tf)

    # Update modified  Cp
    return cp * e + c;

# Model set
def modelSet(c, d, data, m, n, a, b):
    # Slope factor
    Tend     = data.iloc[H - L - 1, n] #(Tend)
    Tini  = data.iloc[0, n]  #(Tinitial)

    # Return list generator
    return [modelPoint(o, c, d,  Tend, Tini, data, m, n, a, b) for o in range(0, H - L)]

# Loss function. Filter for Temperatures around the glass transition
def p(o, n, Tmin, Tmax, data):
    return 1.0 if data.iloc[o, n] < Tmin or data.iloc[o, n] > Tmax else 0.0

# This method returns the vector of residuals for the given parameters.
def residuals(theta, data, m, n, a, b, Tmin, Tmax):
    c   = theta[0]
    d   = theta[1]

    # Starting values for MSE and DeltaH is by default, 0.
    model = modelSet(c, d, data, m, n, a, b)
```

```
105     diffs = [0] * len(model)
106 
107     for o, point in enumerate(model, start=0):
108 
109          # This refers to THE SAME REFERENCE deltaCp that would be in the second column
110         residual   = point - data.iloc[o, 2]
111         diffs[o]   = p(o, n, Tmin, Tmax, data) * residual;
112 
113     return diffs
114 
115 # Calculation of the recovered enthalpy
116 def deltaH(c, d, data, m, n, a, b, Tmin, Tmax):
117     dH = 0
118     model = modelSet(c, d, data, m, n, a, b)
119     for o in range(0, len(model) - 1):
120 
121         residual   = model[o] - data.iloc[o, 2]
122         inverseP   = 1.0 - p(o, n, Tmin, Tmax, data)
123         dH         = dH + inverseP * (residual * (data.iloc[o + 1, n] - data.iloc[o, n]))
124     return dH
125 
126 resultSet = pd.DataFrame({'TRef': data.iloc[:-1, 1], 'CPRef':data.iloc[:-1, 2]})
127 
128 # TO SAVE A FIGURE OF ALL THE ANALYZED DATA: creation of the figure with size and resolution
129 figure(figsize=(15, 11), dpi=240)
130 
131 # Now we start the analysis of all the measurements (starting from the 4th column
132 #until all the aged experiments)
133 for col in range(3, Ncols, 2):
134     # Initial time
135     initTime = datetime.now().time()
136     initTimeseconds = (initTime.hour * 60 + initTime.minute) * 60 + initTime.second
137 
138     # Initial Cp_Aged in the m column, for the particular aging time you desire
139     m = col + 1;
140     # Related Temperature should be to the left of Cp data
141     n = m - 1;
142 
143     name = data.columns.values[col+1]
144     I.append(name)
145 
146     #a and b represent exact Cp cell values
147     a = data.iloc[0, m]; #Cp(T=Tini)
148     b = data.iloc[H - L - 1, m]; #Cp(T=Tend)
149 
```

```
    # Least Square fitting to find c and d
    result = least_squares(residuals,
                           theta0, # initial guess at starting point
                           args = (data, m, n, a, b, Tmin, Tmax), # alternatively you can do this
                          #with closure variables in f if you like
                           ftol=None,
                           gtol=1e-15,
                           xtol=None,
                           max_nfev=200,  #number of iterations to obtain c and d.
                          )

    c = result.x[0]
    d = result.x[1]
    C.append(c)
    D.append(d)

    resultDf = modelSet(c, d, data, m, n, a, b)     #CP CORRECTED
    CHI.append(sum([x ** 2 for x in result.fun]))  # CALCULATION OF THE MSE

    # Now we calculate the linear fits for the glass and the liquid state
    #Linear fit of the glassy state for  T<Tglass
    x = []
    for o in range(0, H - L):
        temp = data.iloc[o, 1]
        if temp > Tglass:
            break
        x.append(data.iloc[o, 1])

    x = np.array(x).reshape(-1, 1)
    y = np.array(data.iloc[0:len(x), 2])
    model = LinearRegression().fit(x, y) #LINEAR FIT OF THE GLASS FOR T<Tglass AND THEN EXTENDED

    #Linear fit of the liquid state for  T>Tmelt
    x = []
    for o in range(0, H - L):
        temp = data.iloc[o, 1]
        if temp > Tmelt:
            x.append(data.iloc[o, 1])

    x1 = np.array(x).reshape(-1, 1)
    y1 = np.array(data.iloc[H-L-len(x1)+1:, 2]) #np.array(data.iloc[H-L-len(x1):len(x1), 2])
    model1 = LinearRegression().fit(x1, y1)  #LINEAR FIT OF THE LIQUID FOR T>TMAX AND THEN EXTENDED

    # Now we calculate the integrals to determine Tf using the matching areas
    x = []
```

```
    iTmax = -1
    iTmin = H-L
    for o in range(0, H - L):
        temp = data.iloc[o, n]
        if temp < Tmelt and temp > Tglass:
            x.append(temp)
            iTmax = max(iTmax, o)
            iTmin = min(iTmin, o)

    # Integral of Cp corrected
    x = np.array(x)
    y = np.array(resultDf[iTmin:iTmax+1])
    Icp = simps(y, x)

    # Integral of the glassy curve
    x0   = np.array(data.iloc[iTmin:iTmax+1, 1])
    x0_t = x0.reshape(-1, 1)

    Iinf = simps(model.predict(x0_t), x0)

    # Integral of the liquid curve
    x0   = np.array(data.iloc[iTmin:iTmax+1, 1])
    x0_t = x0.reshape(-1, 1)

    Isup = simps(model1.predict(x0_t), x0)

    # Integral of Cp corrected with respect of the glassy curve from Tmin to Tmax
    AreaCp = Icp - Iinf

    diff = float("inf")
    bestTf = 0

    #The error in Tf depends of this discretization of this variation. IN THIS CASE: 0.005
    for Tf in np.arange(Tmin, Tmax, 0.005): #HERE WE VARY TF
    #AND CALCULATE THE AREAS TO OBTAIN THE TF WHERE THEY MATCH
        x0   = np.array(np.arange(Tf, Tmax, 0.1))
        x0_t = x0.reshape(-1, 1)

        Isup2 = simps(model1.predict(x0_t), x0)
        Iinf2 = simps(model.predict(x0_t), x0)

        #Now we determine the value of Tf according to the matching areas
        #Here we minimize Eqn. 7
        if abs(Isup2 - Iinf2 - AreaCp) < diff:
            diff = abs(Isup2 - Iinf2 - AreaCp)
```

```
            bestTf = Tf
            #Values of Tf that minimized Eqn. 7

    TF.append(bestTf)

    DH.append(deltaH(c, d, data, m, n, a, b,Tmin, Tmax))

    #End Time
    endTime = datetime.now().time()
    endTimeseconds = (endTime.hour * 60 + endTime.minute) * 60 + endTime.second
    totalTime = endTimeseconds - initTimeseconds
    RUNNINGTime.append(totalTime)

    #Calculation of Delta Cp
    differencia = resultDf - data.iloc[:-1, 2]

    #Saving the Corrected Cp and Delta Cp
    resultSet[f'TCor_{name}(K)']  = data.iloc[:-1, n]
    resultSet[f'CPCor_{name}(J/gK)'] = resultDf
    resultSet[f'DeltaCp_{name}(J/gK)'] = differencia

    #Plotting the Corrected Cp for each aged column.
    #If you don't want to plot comment this lines (the program will be faster)
    plt.grid()
    plt.plot(data.iloc[:-1, 1], resultDf,  linewidth=2, label=f'Aged PS q(K/s) = {name}')
    # plot all Cp(T) corrected
    x0 = np.array(data.iloc[:-1, n]).reshape(-1, 1)
    plt.xlabel('T(K)')
    plt.ylabel('$C_p (J/g K)$')

#Continuation of the same plot with the Reference and the liquid and glass lines
plt.plot(data.iloc[:-1, 1], data.iloc[:-1, 2], linewidth=2, label='Reference PS')
# cp reference ONLY ONE REFERENCE
x0 = np.array(data.iloc[:-1, n]).reshape(-1, 1)
plt.plot(x0, model.predict(x0), linewidth=2, label='Glass Line')  #glassy state fit
plt.plot(x0, model1.predict(x0), linewidth=2, label='Liquid Line' )  #liquid state fit
plt.title("Method: PA") #title of the plot
plt.xlabel('T(K)') #name of x-axes
plt.ylabel('$C_p (J/g K)$') #name of y-axes
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')  #location of the legand
plt.tight_layout()

#Name and location of the output figure to be saved
plt.savefig(f"C:/Users/.../graph.png", dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', format=None,
```

```
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None, metadata=None)

plt.show()   #Here we show the graph

#Location of the outpit Excel file for the corrected Cp and Delta Cp vs T
resultSet.to_excel(f"C:/Users/.../CORRECTED-Cp.xlsx", index=False)

#Saving all the calculated parameters
resultSet = pd.DataFrame({'v(K/s) or t(s)': I[:], 'c': C[:], 'd': D[:], 'MSE': CHI[:],
'Tf(K)': TF[:],'Dh(J/g)': DH[:],'running_time(s)': RUNNINGTime[:]})

#Location of the output Excel file for all the calculated parameters
resultSet.to_excel(r"C:\Users\...\parameters.xlsx", index=False)
```

Fig. 2 shows a typical graph output of $c_p(T)$ vs. T for one aged sample and its reference. Also all the input values of the temperatures used are shown. It's possible to observe the glass and the liquid lines and that the corrected $c_p^{agedOTP}$ of the the aged sample match with the reference c_p^{refOTP} below and after the glass transition.

Fig. 3 shows an screenshot of the Excel output file of the code. The first two columns correspond to the temperature and the c_p^{ref} of the reference and then the rest of the columns are the temperatures and the c_p^{aged} and the Δc_p of the corrected aged curves.

Finally, Fig. 4 shows a screenshot of the Excel output file of the obtained parameters for the different aged experiments performed. In this case, for different cooling rates q. The output parameters are: the values of c (a) and d (b), the MSE obtained from the correction of the data (vertical shifts and slope changes), the obtained value of T_f in K, the ΔH in J/g and the running time for each aged sample in seconds.

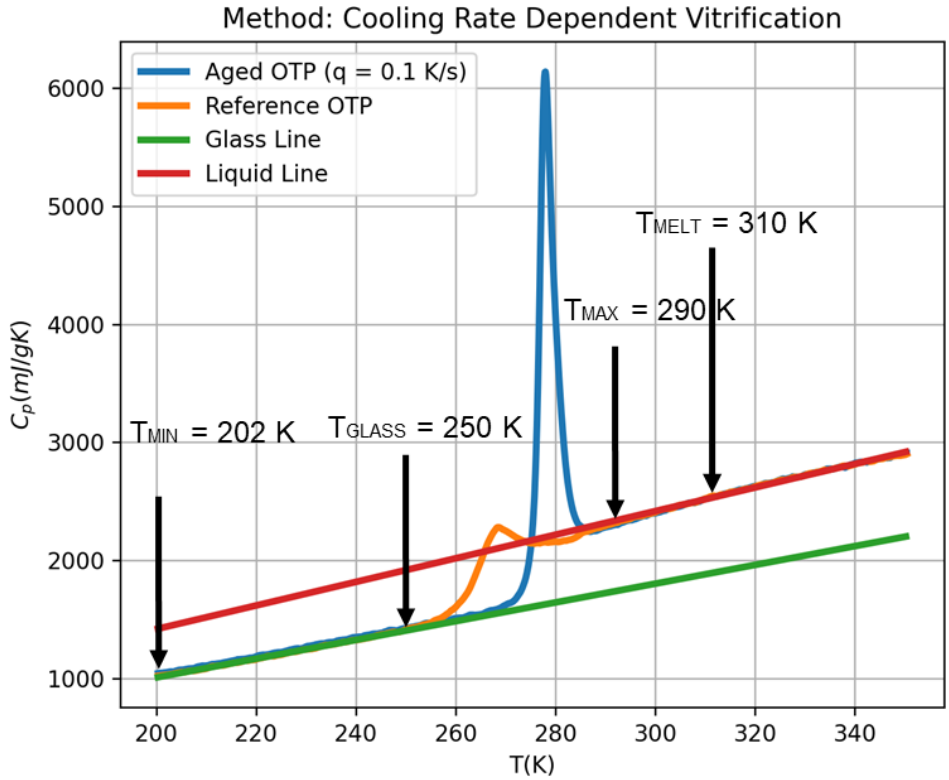
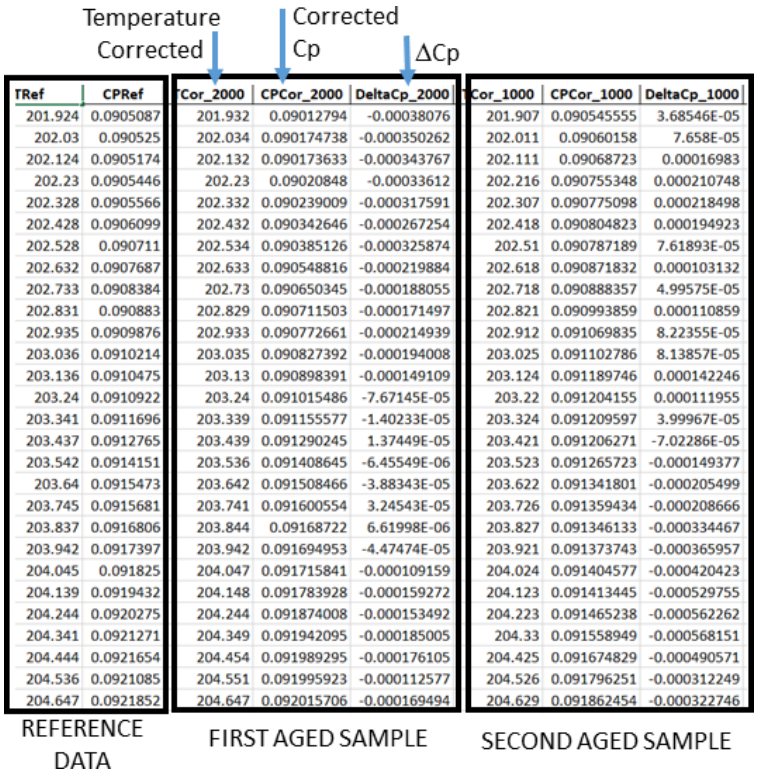


Figure 2: Output graph of c_p vs T of the machine learning based method in Phyton for an aged sample of OTP after cooling at 0.1 K/s (blue lines) and the reference (organge lines). Note also the lines of the glass and liquid as obtained from the linear fits automatically done by this Method. The figure also show the input parameters used to run the code.

B) Method to obtain Ton from FSC Measurements

This Method was designed to obtain the value of T_{onset} (T_{ON}). We recommend to run this method before running VITRIFAST. For this method we only need the data of the most aged sample of the set to be analyzed. So the input of this code are two columns that correspond to the reference data (T and c_p^{ref} or Heat Flow), and other two columns that correspond to the most aged sample (T and c_p^{aged} or Heat Flow). The input parameters to run this code are listed below and showed in Fig. 5. The obtained value of T_{ON} will be later used as an input in the program VITRIFAST in the place of T_{min} and will be the same for the analysis of all the data set. It's important to note that that aslo it's possible to not to run this Method and select he value of T_{ON} (T_{min}) by hand in the VITRIFAST program.



TRef	CPRef	TCor_2000	CPCor_2000	DeltaCp_2000	TCor_1000	CPCor_1000	DeltaCp_1000
201.924	0.0905087	201.932	0.09012794	-0.00038076	201.907	0.090545555	3.68546E-05
202.03	0.090525	202.034	0.090174738	-0.000350262	202.011	0.09060158	7.658E-05
202.124	0.0905174	202.132	0.090173633	-0.000343767	202.111	0.09068723	0.00016983
202.23	0.0905446	202.23	0.09020848	-0.00033612	202.216	0.090755348	0.000210748
202.328	0.0905566	202.332	0.090239009	-0.000317591	202.307	0.090775098	0.000218498
202.428	0.0906099	202.432	0.090342646	-0.000267254	202.418	0.090804823	0.000194923
202.528	0.090711	202.534	0.090385126	-0.000325874	202.51	0.090787189	7.61893E-05
202.632	0.0907687	202.633	0.090548816	-0.000219884	202.618	0.090871832	0.000103132
202.733	0.0908384	202.73	0.090650345	-0.000188055	202.718	0.090888357	4.99575E-05
202.831	0.090883	202.829	0.090711503	-0.000171497	202.821	0.090993859	0.000110859
202.935	0.0909876	202.933	0.090772661	-0.000214939	202.912	0.091069835	8.22355E-05
203.036	0.0910214	203.035	0.090827392	-0.000194008	203.025	0.091102786	8.13857E-05
203.136	0.0910475	203.13	0.090898391	-0.000149109	203.124	0.091189746	0.000142246
203.24	0.0910922	203.24	0.091015486	-7.67145E-05	203.22	0.091204155	0.000111955
203.341	0.0911696	203.339	0.091155577	-1.40233E-05	203.324	0.091209597	3.99967E-05
203.437	0.0912765	203.439	0.091290245	1.37449E-05	203.421	0.091206271	-7.02286E-05
203.542	0.0914151	203.536	0.091408645	-6.45549E-06	203.523	0.091265723	-0.000149377
203.64	0.0915473	203.642	0.091508466	-3.88343E-05	203.622	0.091341801	-0.000205499
203.745	0.0915681	203.741	0.091600554	3.24543E-05	203.726	0.091359434	-0.000208666
203.837	0.0916806	203.844	0.09168722	6.61998E-06	203.827	0.091346133	-0.000334467
203.942	0.0917397	203.942	0.091694953	-4.47474E-05	203.921	0.091373743	-0.000365957
204.045	0.091825	204.047	0.091715841	-0.000109159	204.024	0.091404577	-0.000420423
204.139	0.0919432	204.148	0.091783928	-0.000159272	204.123	0.091413445	-0.000529755
204.244	0.0920275	204.244	0.091874008	-0.000153492	204.223	0.091465238	-0.000562262
204.341	0.0921271	204.349	0.091942095	-0.000185005	204.33	0.091558949	-0.000568151
204.444	0.0921654	204.454	0.091989295	-0.000176105	204.425	0.091674829	-0.000490571
204.536	0.0921085	204.551	0.091995923	-0.000112577	204.526	0.091796251	-0.000312249
204.647	0.0921852	204.647	0.092015706	-0.000169494	204.629	0.091862454	-0.000322746

Figure 3: Screenshot of the Excel Output File: CORRECTED-Cp.xlsx. The first two columns correspond to the T and the c_p of the reference, and then the other columns correspond to the T_{aged}, c_p^{aged} and Δc_p^{aged} of the "n" aged samples analyzed.

The input parameters to run this code to determine T_{ON} are:

L: the value of the row (the lowest index) to consider the data to analyze.

H: the value of the row (the higher index) to consider the data to analyze.

T_{min}: The value of the temperature **just below** the glass transition starts. It's important that this value is the maximal possible value of T before the transition starts because this program will analyze all the possibles values of T_{ON} until this value of T_{min}.

T_{max}: the value of the temperature to determine the end of the glass transition.

The output of this Method are two possibles values of T_{ON} that will be found printed in the box below the code after running it. Those values of T_{ON} are obtained by two different methods: one method consist on determining at which temperature the area between the c_p^{aged} corrected curve and the glass line is maximal. For that, the program calculates all the possibles values of that area changing the initial value of the integration: from the one

v(K/S)	c	d	MSE	Tf(K)	Dh(J/mol)	running_time(s)
2000	-0.00035	0.090134	0.000159	261.6	0.54366006	8
1000	-0.0011	0.091025	0.00012	261	3.464122382	7
500	-0.00207	0.091541	0.000172	260	8.749181744	7
300	-0.00258	0.091069	0.000256	258.89	14.29144317	7
200	-0.00282	0.091605	0.000281	258	18.32848564	7
100	-0.00289	0.091378	0.000273	257.31	21.60741242	7
50	-0.00297	0.091784	0.000244	256.2	27.44036581	6
30	-0.00327	0.092247	0.000243	255	33.54104421	7
20	-0.00316	0.091979	0.000282	254.1	37.04400386	7
10	-0.0034	0.092368	0.000296	253.4	40.88627475	10
5	-0.00323	0.092155	0.000254	252.105	46.01190269	7
3	-0.00352	0.091909	0.00026	251.005	51.33837646	9
2	-0.00339	0.09189	0.000317	250.3	54.31133531	8
1	-0.00339	0.091848	0.000251	249.8	57.27176621	9
0.5	-0.00342	0.092529	0.000344	248.6	61.75963777	9
0.3	-0.00343	0.092096	0.00034	247.4	66.56514714	8
0.2	-0.00336	0.092293	0.000358	247	68.70662925	8
0.1	-0.00332	0.092199	0.000375	245.47	74.16393013	7

Figure 4: Screenshot of an Excel Output File: parameters.xlsx of VITRIFAST program. The first column save the value of the header of the input file that characterized each aged sample. Then the other columns show the calculated parameters: $c(a)$, $d(b)$, MSE, T_f, ΔH and the running time. Each lines correspond to each aged sample.

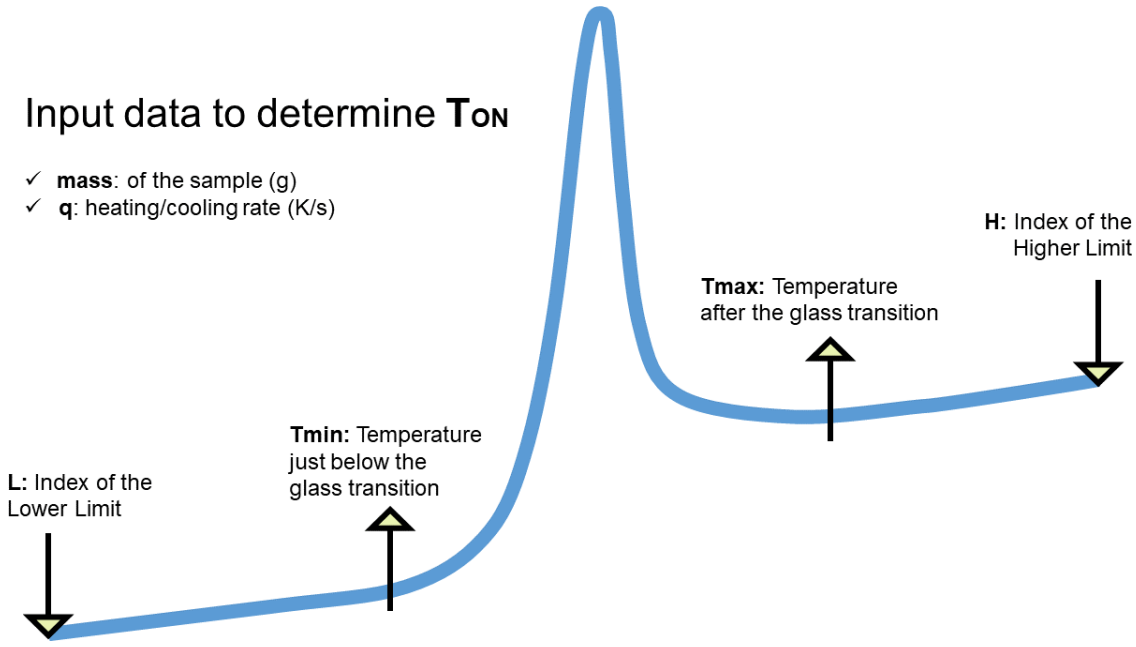


Figure 5: Input parameters to run this method to determine T_{ON}. The obtained value of T_{ON} will be used by VITRIFAST as an input in the place of T_{min}.

corresponding to the lowest value of the data (index L) and the one of the highest value (corresponding to the value set as T_{min}). One we have all the possibles areas, we select the

highest value of the area, the maximal area, then the value of the T_{ON} correspond to the lowest value of temperature where the integration started.

The second Method to determine T_{ON} is the opposite idea of the first one. In this case what we do is to minimize the area between the glass line and the c_p curve for temperatures below the value of T_{ON}. We calculate all the possible areas values and then we keep the lowest. In the best scenario should be obtain the same value of T_{ON} for both Methods. If that is not the case, it's up to the user to decide whether to use the result of one Method or the other or an average between them.

```
import pandas as pd
from scipy.optimize import least_squares
import numpy as np
from sklearn.linear_model import LinearRegression
from scipy.integrate import simps
import matplotlib.pyplot as plt
from datetime import datetime

#RUN THIS METHOD TO DETERMINE THE VALUE OF T_ON. FOR THAT USE A VERY AGED SAMPLE.

#What is needed is a spreadsheet where the first two columns are the Temp/reference Cp data
#and the other two columns are the data with more aging: Temp/More_Aged_Cp data
filename = 'Data.xlsx'

#INPUT, as necessary based on the sheet called
sheet = 0

#LOAD DATA FROM INPUT
data = pd.read_excel(filename, sheet_name=sheet, header=0)

#INPUT, these should be the same for one set of aging experiments
#Lower and  Higher bounds of  the experiment
L = 260   #Lower limit
H = 1990  #higher limit

#Tmin and Tmax represent the filter in temperature before and after glass transition
Tmin = 252  #Tmin MUST BE JUST BELOW the glass transition TO OBTAIN PROPERTLY T_ON.
Tmax = 331   #Tmax IS AFTER the glass transition, EXAMPLE Tf + 50K

#Initial values of c and d-> both in zero
c = 0.0
```

```
d = 0.0
theta0 = [c, d]

# Slice the data within the range
data = data[L:H+1].reset_index()

#Number of columns to analyze(this case only 4)
Ncols = 4
col = 3

# Aged Cp column of interest INPUT,
m = col + 1;
# Related Temperature should be to the left of Cp data
n = m - 1;

#Initial time for this Method
initTime = datetime.now().time()
initTimeseconds = (initTime.hour * 60 + initTime.minute) * 60 + initTime.second

#Conversion from Celcius to Kelvin. You can change that offset to do another conversion.
tempOffset = 273.15
tempCols = [x for x in range(1, Ncols+1, 2)]
data.iloc[:, tempCols]  = data.iloc[:, tempCols] + tempOffset

#Conversion from HF to CP (multiply for DeltaCP at Tg in DSC measure)
DeltaCPatTG = -1 # J/K*mol
CpCols = [x for x in range(2, Ncols, 2)]
data.iloc[:, CpCols]  = data.iloc[:, CpCols] * DeltaCPatTG

#a and b represent exact Cp cell values
a = data.iloc[0, m]; #Cp(T=Tini)
b = data.iloc[H - L - 1, m]; #Cp(T=Tend)

# Creation of the variables we want to save
CHI = []
C = []
D = []
TF = []
I = []
DH = []
RUNNINGTime = []
name = data.columns.values[col+1]
I.append(name)

#What is needed is a spreadsheet where the first two columns are the Temp/reference Cp data
```

```
#and the remaining are Temp/Aged Cp data in alternating form

# Model used
def modelPoint(o, c, d, Tend, Tini, data, m, n, a, b):
    cp    = data.iloc[o, m]  #cp(T)

    DeltaT = Tend - Tini #Delta_T
    coef   = (1.0 - (d / a)) / DeltaT
    e      = ( coef *  cp + b - Tend * coef ) / b;

    # Update modified  Cp
    return cp * e + c;

# Model set
def modelSet(c, d, data, m, n, a, b):
    # Slope factor
    Tend     = data.iloc[H - L - 1, n] #(Tend)
    Tini  = data.iloc[0, n]  #(Tinitial)

    # Return list generator
    return [modelPoint(o, c, d,  Tend, Tini, data, m, n, a, b) for o in range(0, H - L)]

# Loss function to reduce the influence of outliers. Filter in the temperatures around the transition
def p(o, n, Tmin, Tmax, data):
    return 1.0 if data.iloc[o, n] < Tmin or data.iloc[o, n] > Tmax else 0.0

# This method returns the vector of residuals for the given parameters.
# Input: theta[0] is c, and X[1] is d.
def residuals(theta, data, m, n, a, b, Tmin, Tmax):
    c   = theta[0]
    d   = theta[1]

    # Starting values for MSE and DeltaH is by default, 0.
    model = modelSet(c, d, data, m, n, a, b)
    diffs = [0] * len(model)

    for o, point in enumerate(model, start=0):

         # This refers to THE SAME REFERENCE deltaCp that would be in the second column
        residual   = point - data.iloc[o, 2]
        diffs[o]   = p(o, n, Tmin, Tmax, data) * residual;

    return diffs

def deltaH(c, d, data, m, n, a, b, Tmin, Tmax):
```

```
    dH = 0
    model = modelSet(c, d, data, m, n, a, b)
    for o in range(0, len(model) - 1):

        residual   = model[o] - data.iloc[o, 2]
        inverseP   = 1.0 - p(o, n, Tmin, Tmax, data)
        dH         = dH + inverseP * (residual * (data.iloc[o + 1, n] - data.iloc[o, n]))
    return dH

resultSet = pd.DataFrame({'TRef': data.iloc[:-1, 1], 'CPRef':data.iloc[:-1, 2]})

result = least_squares(residuals,
                       theta0, # initial guess at starting point
                       args = (data, m, n, a, b, Tmin, Tmax),
                       # alternatively you can do this with closure variables in f if you like
                       ftol=None,
                       gtol=1e-15,
                       xtol=None,
                       max_nfev=200,  #number of iterations to obtain c and d.
                      )

c = result.x[0]
d = result.x[1]
C.append(c)
D.append(d)

resultDf = modelSet(c, d, data, m, n, a, b)    #CP CORRECTED AGED
CHI.append(sum([x ** 2 for x in result.fun]))  # CALCULATION OF THE CHI SQUARE

# NOW WE DO THE LINEAR FITS OF THE GLASS AND LIQUID CURVES UNTIL WE GET THE OPTIMAL VALUE OF TON
cps = resultDf
temps = np.array(data.iloc[:-1, n])

# WE START FROM THE ELEMENT 5TH OF ALL OUR DATA
lowerBoundIdx = 5
MaxArea = -np.inf
MinArea = np.inf

TmaxIdx = np.inf
TminIdx = np.inf

for idx, temp in enumerate(temps):
    if temp > Tmax:
        TmaxIdx = min(TmaxIdx, idx)
    if temp > Tmin:
```

```
        TminIdx = min(TminIdx, idx)

# WE EVALUATE DIFFERENT VALUES OF TON
for TonIdx in range(lowerBoundIdx, TminIdx):
    x = np.array(temps[:TonIdx]).reshape(-1, 1)
    y = np.array(cps[:TonIdx])
    model = LinearRegression().fit(x, y) #LINEAR FIT OF THE GLASS FOR T<TON AND THEN EXTENDED

    x = np.array(temps[TonIdx:TmaxIdx])
    y = np.array(cps[TonIdx:TmaxIdx])
    IcpRight = simps(y, x)    # INTEGRAL CP CORRECTED

    x0   = np.array(temps[TonIdx:TmaxIdx])
    x0_t = x0.reshape(-1, 1)
    IinfRight = simps(model.predict(x0_t), x0)

    AreaCpRight = abs(IcpRight - IinfRight)
    #INTEGRAL CP WITH RESPECT TO GLASS CURVE FROM TON TO TMAX

    x = np.array(temps[:TonIdx])
    y = np.array(cps[:TonIdx])
    IcpLeft = simps(y, x)    # INTEGRAL CP CORRECTED BELOW TON

    x0   = np.array(temps[:TonIdx])
    x0_t = x0.reshape(-1, 1)
    IinfLeft = simps(model.predict(x0_t), x0)    # INTEGRAL LINEAR FIT GLASS BELOW TON

    AreaCpLeft = abs(IcpLeft - IinfLeft) #INTEGRAL CP WITH RESPECT TO GLASS CURVE


    if MaxArea < AreaCpRight:   #OBTAIN THE MAXIMAL INTEGRAL FOR T > TON
        MaxArea = AreaCpRight
        MaxTonIdx = TonIdx

    if MinArea > AreaCpLeft:    #OBTAIN THE MINIMAL INTEGRAL FOR T < TON
        MinArea = AreaCpLeft
        MinTonIdx = TonIdx

print(temps[MaxTonIdx])   #OBTAINED TON METHOOD 1
print(temps[MinTonIdx])   #OBTAINED TON METHOOD 2
```

C) Phyton Code used to include Noise to the FSC Measurements

This code was employed to add noise to the data in order to test the robustness of the Method VITRIFAST and determine until which levels of noise are acceptable in order to trust the results of this method. We added white noise to a standard data set obtained via FSC defining the noise ratio as δ:

$$\delta = \frac{\varepsilon}{\Delta c_p} \tag{1}$$

where ε is the amplitude of the added random white noise to the specific heat curves with an amplitude of approximately $\sim 3\sigma$, and $\Delta c_p = c_p^{liquid}(T_{max}) - c_p^{glass}(T_{min})$ is the difference of the specific heat in the liquid and the glassy state, see Fig. 6.

Note that the values of δ varies from zero noise to the maximal value of the unity, for which $\varepsilon = \Delta c_p$, making impossible to determine precisely the glass transition. We tested our ML-based method including the noise δ in two measures after cooling at 2000 K/s and at 0.1 K/s.

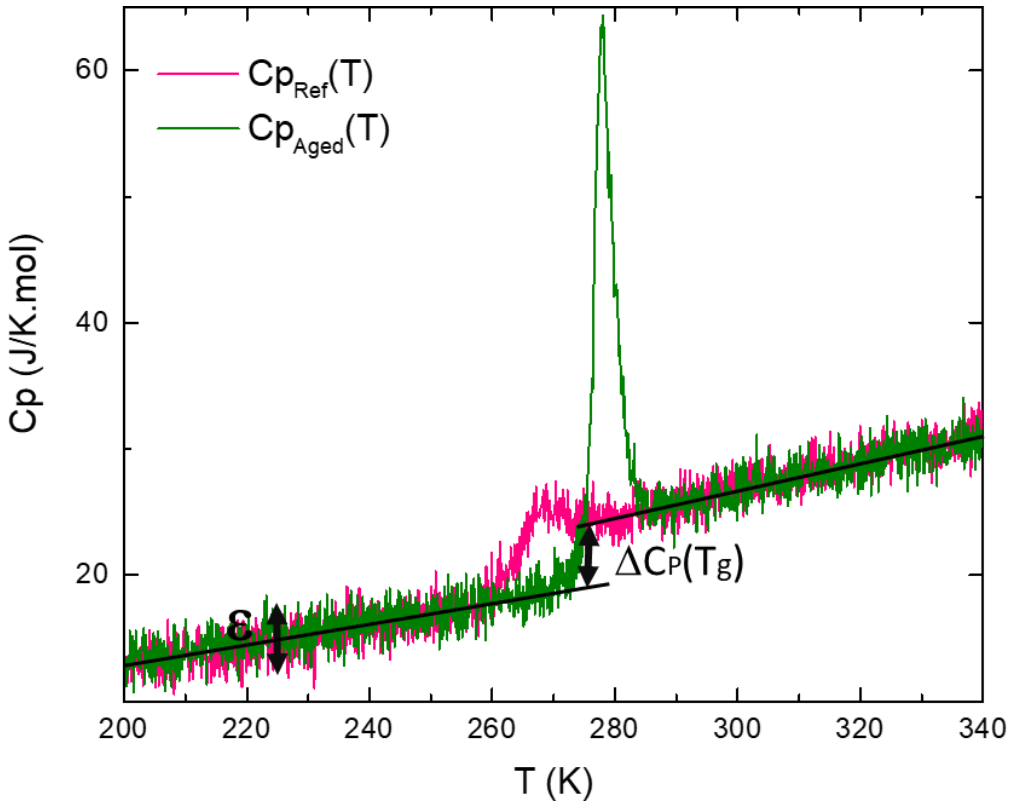


Figure 6: Curves of $c_p(T)$ for the aged and reference measures via FSC with an addition of white noise, ε, with an amplitude of 3σ of it's gaussian distribution from random generator. We used this curves as an input to determine the robustness of this ML-based method upon noise.

```
import numpy as np
import matplotlib.pyplot as plt
import statistics

Ref_watts = data.iloc[:-1, 2]  ** 2
resultDf_watts = data.iloc[:, m]  ** 2

Ref_dB = 10 * np.log10(Ref_watts)
resultDf_dB = 10 * np.log10(resultDf_watts)

# Calculate signal power and convert to dB
ref_avg_watts = np.mean(Ref_watts)
sig_avg_watts = np.mean(resultDf_watts)

sig_avg_db_ref = 10 * np.log10(ref_avg_watts)
sig_avg_db = 10 * np.log10(sig_avg_watts)
```

```
17
18  mean_noise = 0
19
20  Delta_resultDf = resultDf[iTmax]  - resultDf[iTmin]
21
22  resultSet1 = pd.DataFrame()
23  # Set different targets of Noise
24  target_snr_db_list = [60, 49, 42, 40, 36, 33, 30, 27, 25, 23, 21, 20, 18, 16]
25  name_list = [0.006, 0.025, 0.05, 0.07, 0.1, 0.15, 0.22, 0.3, 0.4, 0.5, 0.6, 0.7, 0.88, 1]
26
27  for idx in range(0, len(target_snr_db_list)):
28
29      target_snr_db = target_snr_db_list[idx]
30      name = name_list[idx]
31
32      noise_avg_db_ref = sig_avg_db_ref - target_snr_db
33      noise_avg_db = sig_avg_db - target_snr_db
34
35      noise_avg_ref = 10 ** (noise_avg_db_ref / 10)
36      noise_avg = 10 ** (noise_avg_db / 10)
37
38      # Generating an sample with white noise
39      noise_ref = np.random.normal(mean_noise, np.sqrt(noise_avg_ref), len(Ref_watts))
40      noise = np.random.normal(mean_noise, np.sqrt(noise_avg), len(resultDf_watts))
41
42      # Adding noise to the Original Signal
43      Ref_noise = data.iloc[:-1, 2]  + noise_ref
44      resultDf_noise = data.iloc[:, m] + noise
45
46      # Calculation delta
47      mean_noise_ref = 3 * statistics.stdev(noise_ref)
48      mean_noise = 3 * statistics.stdev(noise)
49
50      delta_noise_ref = mean_noise_ref/Delta_resultDf
51      delta_noise = mean_noise/Delta_resultDf
52
53      # Saving all the noisy data, for the reference and the aged sample with the same noise
54      resultSet1[f'TRef_Noise_{name}']  = data.iloc[:-1, 1]
55      resultSet1[f'CPRef_Noise_{name}'] = Ref_noise
56      resultSet1[f'TCorr_Noise_{name}']  = data.iloc[:, n]
57      resultSet1[f'CPCorr_Noise_{name}'] = resultDf_noise
58
59  #Folder to save the data
60  resultSet1.to_excel(f"C:/Users/Noisy-Data.xlsx", index=False)
61
```

```
#Plotting Noisy Data
plt.plot(data.iloc[:, n], resultDf_noise)
plt.plot(data.iloc[:-1, 1], Ref_noise)
plt.show()
```

D) Parameters employed to analyze the examples showed in the main text

In this work we analyzed two materials: OTP and PS. For the conversion from Heat Flow (HF) to units of c_p we employed the following equation:

$$Cp[J/g.K] = \frac{HF[J/s]}{m[g].q[K/s]} \tag{2}$$

where m is the mass and q is the cooling rate used to measure. For the determination of the mass in our experiments we consider the following equation:

$$m[g] = \frac{\Delta HF(T_g)}{\Delta c_p(T_g).q} \tag{3}$$

where $\Delta HF(T_g)$ is the step in HF at T_g and $\Delta c_p(T_g)$ is the step obtained from a DSC measurements in the literature. In the case of OTP we consider that Δc_p (T_g = 240 K) = 0.486 J/g.K and in the case of PS we consider that Δc_p (T_g = 375 K) = 0.292 J/g.K . Considering those values we obtained the mass of the both samples OTP and PS showed in Table S1 and then we could perform the conversion from HF to c_p automatically in the VITRIFAST program.

Material	q [K/s]	mass [ng]	Tmin [K]	Tmax [K]	Tglass [K]	Tmelt [K]	L	H
OTP	1000	88	202	290	250	295	5	1500
PS	1000	160	275	445	360	450	465	2650

Table S1: Parameters used for the analysis of the samples.

Using the parameters listed in the Table S1 as an input, we obtained the following curves

for the aged OTP and PS with the two methods analyzed: physical aging and different cooling rates. Figs. (7-9) show the corrected curves and the corresponding glass and liquid lines obtained.

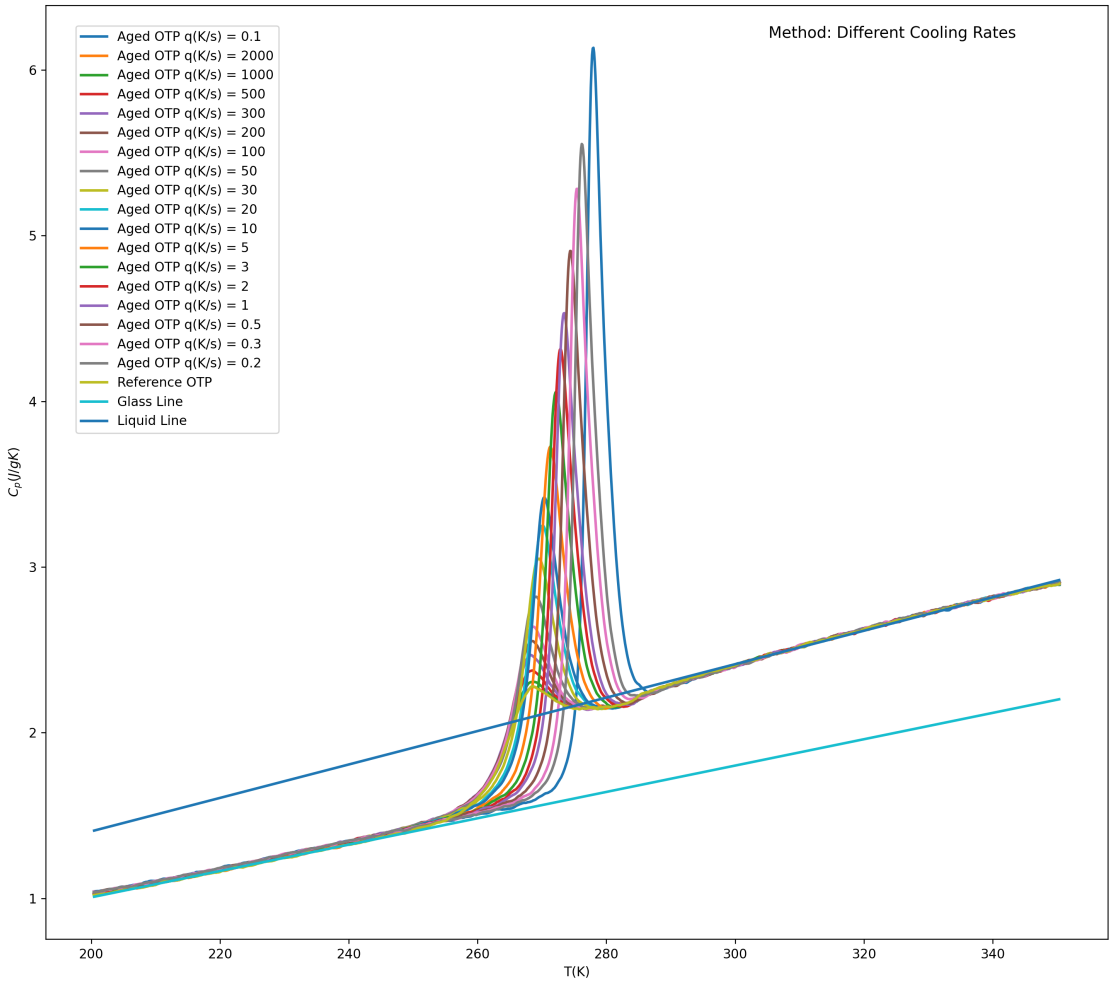


Figure 7: Output for the OTP aged at different cooling rates.

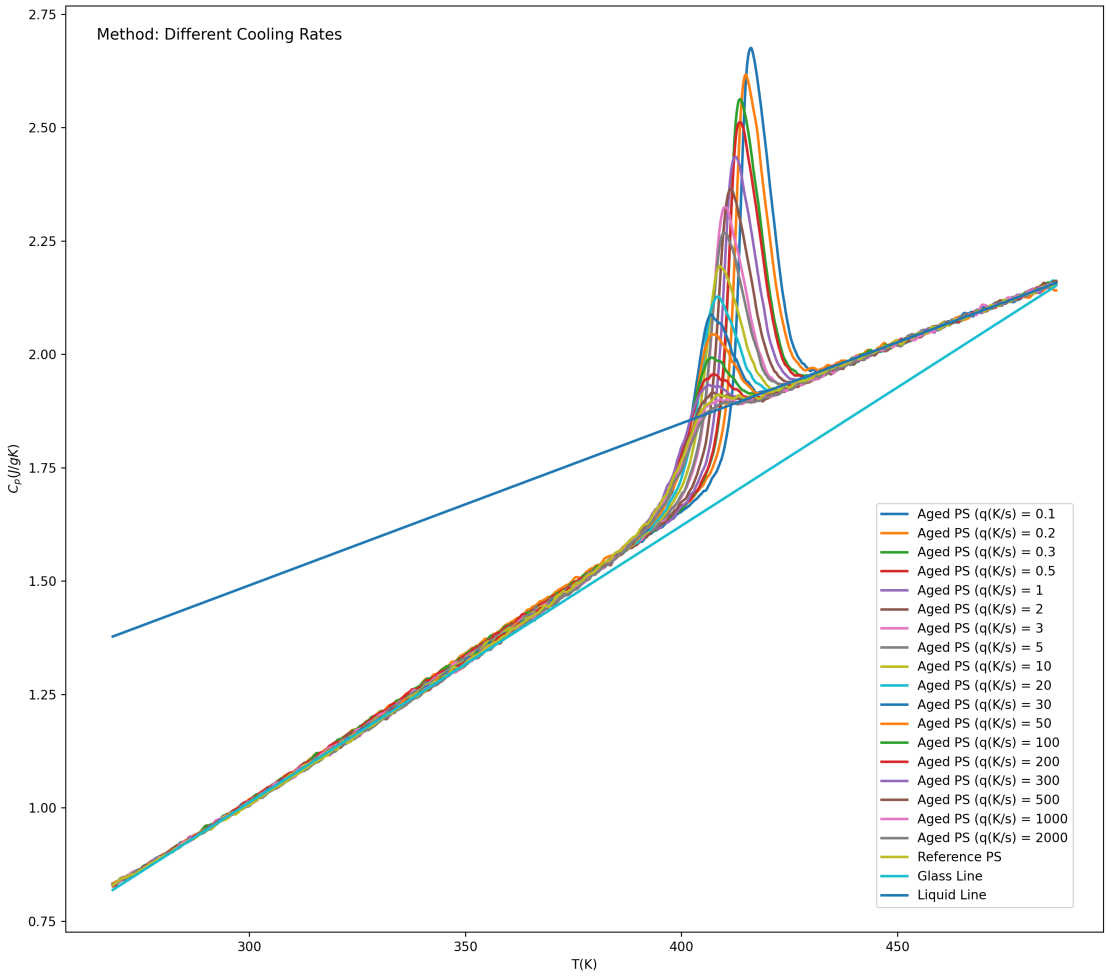


Figure 8: Output for the PS aged at different cooling rates.

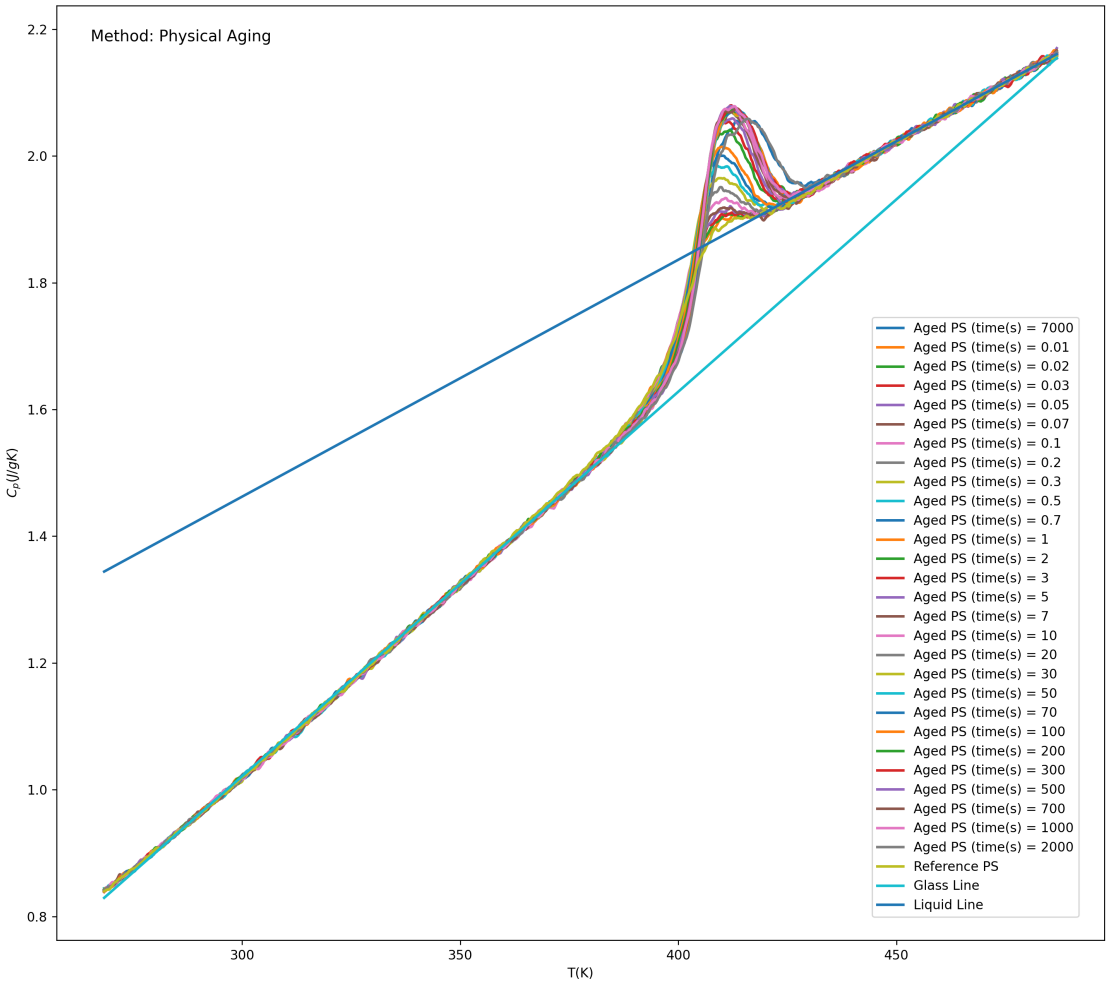


Figure 9: Output for the PS physical aged for different periods of time.