SUPPLEMENTARY MATERIAL FOR : Thermal weakening friction and seismic slip: an efficient numerical scheme for thermal diffusion

S. Nielsen[1], E. Spagnuolo[2], M. Violay[3]and G. Di Toro[4]

[1]Department of Earth Sciences, Durham University, *DH1 3LE* Durham, United Kingdom
[2]Istituto nazionale di Geofisica e Vulcanologia, Via di Vigna Murata, 605, *00143* Roma, Italy
[3]EPFL ENAC IIC LEMR, GC D1 401 (BÃ¢timent GC), Station 18, *CH-1015* Lausanne, Switzerland
[4]Dipartimento di Geoscienze, via G. Gradenigo, 6, *35131* Padova, Italy

Additional tests of friction versus experimental data

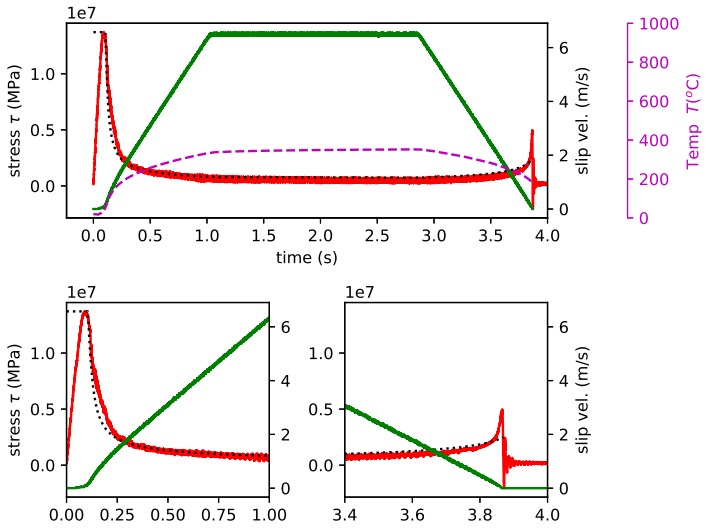


Fig. S1. Experimental fit of friction in Carrara marble with Arrhenius thermal weakening and heat sinks (experiment s308, normal stress 20 MPa). Solid red: measured experimental shear stress. Dotted black: model friction. Solid green: measured slip velocity. Dashed purple: computed temperature.

Corresponding author: Stefan Nielsen, stefan.nielsen@durham.ac.uk

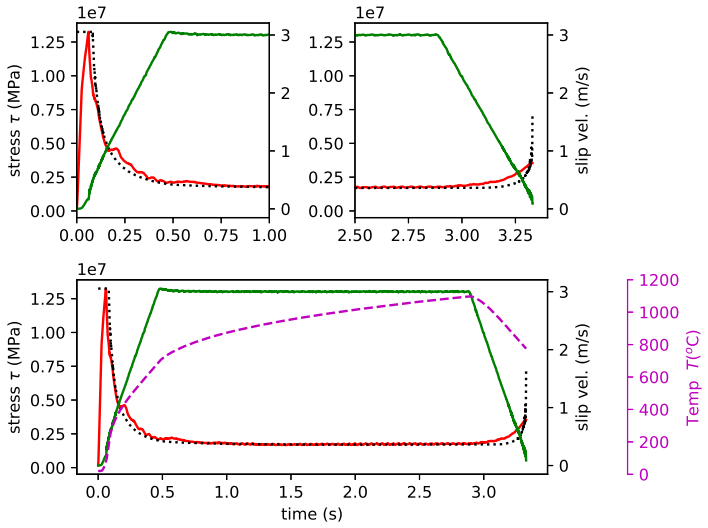


Fig. S2. Experimental fit of friction in gabbro with FWSS (normal stress 20 MPa). Solid red: measured experimental shear stress. Dotted black: model friction. Solid green: measured slip velocity. Dashed purple: computed temperature. In this example it can be seen that the recovery is not well reproduced.

NUMERICAL CODE FOR TEMPERATURE (PYTHON):

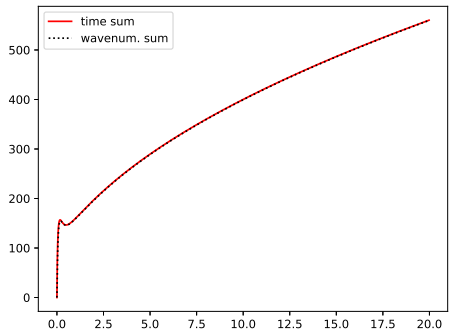
Efficient wavenumber summation example:

```
# INITIALIZATIONS:
import numpy as np
import matplotlib.pyplot as plt
import time as tm
%matplotlib notebook
#
ttot=20.;
dt=ttot/1000;
nt=int(ttot/dt)
rho = 3000; c = 715; k = 1.1e-6;
M = 32
zmax =  (M/2.5) * np.sqrt(k * ttot);
zmin = (1/2) * zmax/M;
smax=2*np.pi/zmin
ds = smax/M;
smin=ds/2
s=np.asarray([(m - 1/2)*ds for m in range(1,M+1)],dtype=float)
theta=np.asarray([0.0 for M in range(0,M)],dtype=float)

# DEFINE THE TEMPERATURE COMPUTATION ROUTINES
# wavenumber summation:
def comp_temp_wav(q, theta):
    qn= q/(rho*c); Twav=0.
    for mm in range(0,M):
        theta[mm]= (dt * qn + theta[mm])/(1.0 + dt * k * s[mm]**2 ) # Back Euler
        Twav=Twav+theta[mm]*ds
    Twav=Twav*(2.0/np.pi)
    return(Twav,theta)
```

```
# classic time summation:
def comp_temp_sum(q):
    Tsum=0.
    for p in range(it):
        Tsum=Tsum + dt*q[p]/np.sqrt(dt * (float(it)-1/4) - dt * p)
    Tsum=Tsum * (1/(2 * rho * c * np.sqrt(k*np.pi)))
    return(Tsum)


# COMPUTE TIME EVOLUTION:
T=[0];Tb=[0];time=[0];theta[:]=0.0;
# imposed heat flow:
q=[5e5+3e6*np.exp(-dt*float(ii)/.1) for ii in range(nt)]
#
# compute with wavenumber summation:
start1=tm.time()
for it in range(1,nt):
    time.append(float(it)*dt)
    Twav,theta=comp_temp_wav(q[it-1]/2., theta)
    T.append(Twav)
end1=tm.time()
## compute with classic time summation (OPTIONAL):
#start2=tm.time()
#for it in range(1,nt):
#    Tsum=comp_temp_sum(q)
#    Tb.append(Tsum)
#end2=tm.time()
#print(end1-start1,end2-start2, (end1-start1)/(end2-start2))
```

Plotting both solutions together:

```
fig,ax=plt.subplots()
ax.plot(time, Tb, 'r-',label='time sum')
ax.plot(time, T,'k:', label='wavenum. sum');
ax.legend()
```



NUMERICAL CODE FOR TEMPERATURE (FORTRAN):

```
c program cotemp2
c computes temperature for an imposed heat flow in 2 different ways
c Stefan Nielsen July 2020
```

```
parameter (M=32,nt=1000)
real*16 k,rho,c,tmax,zmax,zmin,smax,smin,ds,dt,qn,Tb,Tbc,T,pi
real*16 theta(M),s(M)
real*16 q(nt)
integer it,mm
c initializations
k=1.1e-6
rho=3000.
c=715.0
pi=3.1416
dt=0.02
tmax=float(nt)*dt
zmax=float(M)*(2./5.)*sqrt(k*tmax)
zmin=zmax/(2.*float(M))
smax=2.*pi/zmin
ds=smax/float(M)
smin=ds/2.
do mm=1,M
    s(mm)=(float(mm)-0.5)*ds
    theta(mm)=0.0
enddo
c pre-compute values of imposed heat flow:
do it=1,nt
    q(it)=3.0e6*exp(-float(it)*dt/.1) + 0.5e6
enddo
c COMPUTING WITH WAVE NUMBER SUMMATION:
open (22,file='tew.csv',form='formatted')
do it=2,nt !start time loop
    qn=q(it-1)/(2*rho*c)
    Tb=0.
    do mm=1,M ! start memory variables loop
        theta(mm)= dt * qn + theta(mm) * (1. - dt * k * s(mm)**2)
        Tb=Tb+theta(mm)*ds
    enddo ! end memory variables loop
    T=(2./pi)*Tb
    write (22,*) float(it)*dt, T
enddo ! end time loop
close (22)
cc COMPUTING WITH CLASSIC TIME SUMMATION (OPTIONAL):
c open (23,file='tes.csv',form='formatted')
c write (23,*) 0.0, 0.0
c cfact=(1./(2. * rho * c * sqrt(k*pi)))
c do it=2,nt ! start time loop
c    Tbc=0.
c    do j=1,it-1 ! summation over past times
c        Tbc=Tbc+cfact*dt*q(j)/sqrt(dt*(float(it)-float(j)-0.25))
c    enddo
c    write (23,*) float(it)*dt, Tbc
c enddo ! end time loop
c close(23)

stop
end
```