

```

1 #####
2 #EUROPA ROCKY INTERIOR STRENGTH
3 #
4 #DESCRIPTION:
5 #This code calculates various properties of the rocky core of Europa to find
6 #the potential for fractures and other geologic constraints.
7 #
8 #USER INTSTRUCTIONS:
9 #Users can change the variables graphoutput or body to determine the output
10 #and scenario used in the calculations.
11 #"graphoutput" is defined on Line 47
12 #"body" is defined on Line 57
13 #
14 #
15 #CODE AUTHORS:
16 #Henry Dawson, Paul Regensburger, Doug Hemingway
17 #
18 #DATE LAST UPDATED:
19 #November 20, 2025
20 #
21 #ASSOCIATED PUBLICATION:
22 #"Little to No Active Faulting Likely at Europa's Seafloor Today"
23 #Paul K. Byrne, Henry G. Dawson, Christian Klimczak, Paul V. Regensburger,
24 #Kelsey T. Crane, Jeffrey G. Catalano, Catherine M. Elder, Bradford J. Foley,
25 #Christopher R. German, Austin P. Green, Douglas J. Hemingway,
26 #Mohit Melwani Daswani, Mark P. Panning, Noah Randolph-Flagg,
27 #Barbara Sherwood Lollar, Philip Skemer, Steven D. Vance,
28 #and Douglas A. Wiens
29 #Nature Communications
30 #Manuscript number: NCOMMS-25-11067B
31 #####
32
33
34 #Europa Pipeline
35 import numpy as n
36 import pylab as p
37 import time
38 import math
39
40
41 ### DESIRED GRAPHICAL OUTPUT
42 #Change the variable "graphoutput" to the value corresponding with the plot you want
43 #Gravity, full body = 1
44 #Pressure, full body = 2
45 #Linear Fault Strength, core = 3
46 #Log Fault Strength, core = 4
47 #Temperature, core = 5
48 #Brittle vs Ductile Differential Strength = 6
49 #Brittle vs Ductile Absolute Strength = 7
50 #No Graphical Output = 8
51
52 graphoutput=6
53 depth = 100 # m depth below seafloor, change to report values at different depth
54
55 ### SELECTION OF MODEL
56 #Change the variable "body" to the value corresponding to the model you want
57 #Earth = 1
58 #Europa = 2
59 #Enceladus = 3
60 #Moon = 4
61
62 body = 2
63
64 if body==1:
65     name = 'Earth'
66     g_surf = 9.8 # m/s^2 surface gravity

```

```

67 rho_core = 10650 # kg/m^3 density of iron core approximation
68 rho_rock = 4500 # kg/m^3 density of mantle rock approximation
69 rho_crust = 2530 # kg/m^3 density of crustal rock approximation
70 rho_hpice = 0
71 rho_water = 1030 #kg/m^3 density of liquid water
72 rho_out = 926.772 #kg/m^3 density of ice Ih outer shell
73 R = 6378000. # m radius of moon
74 Docean = 3682. # m thickness of liquid ocean
75 Dice = 0. # m thickness of ice shell
76 Dhpipe = 0 # m thickness of high pressure ice
77 Dcrust = 0 # m thickness of lower density crust
78 Riron = 3486000 # m radius of iron core
79 mu_f = 0.15 # friction coeff (Chrysotile, Tesei et al., 2018)
80 #mu_f = 0.000035 # friction coeff (Used for mantle convection)
81 q = 0.02 # K/m
82 strain = 10 ** (-15) # s^-1
83 graph_depth = 500 # Depth shown on plots 3-6
84
85 # #Dry Olivine (Hirth and Kohlstedt, 2003)
86 A = 110000 # MPa^(-3) s^(-1)
87 m = 3.5 # stress exponent
88 Qc = 530000. # J/mol
89 C_OH = 1 #H/10^6 Si, H contentration
90 r_C_OH = 1 # Coefficient on C_OH
91 Volume = 14e-6 # Specific volume for dislocation of dry olivine (Kurato and Rubie,
1997)
92 j = 0 #Grain size exponent
93
94 # #Wet Olivine (Hirst and Kohlstedt, 2003)
95 # A = 89 # MPa^(-3) s^(-1)
96 # m = 3.5 # stress exponent
97 # Qc = 480000. # J/mol
98 # j = 0 #Grain size exponent
99 # C_OH = 1000 # H/10^6 Si (Bai and Kohlstedt, 1992)
100 # r_C_OH = 1.2 # Coefficient on C_OH
101 # Volume = 11e-6 # Specific volume for dislocation of wet olivine
102
103 elif body==2:
104     name = 'Europa'
105     eccen = 0.009
106     g_surf = 1.315 # m/s^2 surface gravity
107     M = 1.89813e27 # mass of Jupiter
108     a = 671100000. # avg orbital radius
109     rho_core = 8000 # kg/m^3 density of iron core approximation
110     rho_rock = 3427 # kg/m^3 density of mantle rock approximation
111     rho_crust = 2530 # kg/m^3 density of crustal rock approximation
112     rho_hpice = 0
113     rho_water = 1025 #kg/m^3 density of liquid water
114     rho_out = 925 #kg/m^3 density of ice Ih outer shell
115     R = 1560800. # m radius of moon
116     Docean = 97000. # m thickness of liquid ocean
117     Dice = 30000. # m thickness of ice shell
118     Dhpipe = 0 # m thickness of high pressure ice
119     Dcrust = 1000 # m thickness of lower density crust
120     Riron = 475000 # m radius of iron core
121     mu_f = 0.15 # friction coeff (Chrysotile, Tesei et al., 2018)
122     q = 0.01 # K/m, temperature gradient
123     strain = 10 ** (-20) # s^-1, strain rate
124     graph_depth = 300 # Depth shown on plots 3-6
125
126
127 # #Wet Olivine (Hirst and Kohlstedt, 2003)
128 A = 89 # MPa^(-3) s^(-1)
129 m = 3.5 # stress exponent
130 Qc = 480000. # J/mol
131 j = 0 #Grain size exponent

```

```

132 C_OH = 100 # H/10^6 Si, H contentration (Bai and Kohlstedt, 1992)
133 r_C_OH = 1.2 # Coefficient on C_OH
134 Volume = 11e-6 # Specific volume for dislocation of wet olivine
135
136 ## OTHER AVAILABLE FLOW LAWS
137
138 # # #Dry Olivine Dislocation (Hirth and Kohlstedt, 2003)
139 # A = 110000 # MPa(-3) s(-1)
140 # m = 3.5 # stress exponent
141 # j = 0 #Grain size exponent
142 # Qc = 530000. # J/mol
143 # C_OH = 1 #H/10^6 Si
144 # r_C_OH = 1 # Coefficient on C_OH
145 # Volume = 14e-6 # Specific volume for dislocation of dry olivine (Kurato and Rubie,
146 1997)
147
148 # # #Wet Olivine Diffusion (Hirth and Kohlstedt, 2003)
149 # A = 2.5e-7 # MPa(-3) s(-1)
150 # m2= 1 # stress exponent
151 # j = 3 #Grain size exponent
152 # Qc = 375000. # J/mol
153 # C_OH = 100 #H/10^6 Si
154 # r_C_OH = 1 # Coefficient on C_OH
155 # Volume = 10e-6 # Specific volume for dislocation of wet olivine (Kurato and Rubie,
156 1997)
157
158 # # #Wet Olivine (Hirst and Kohlstedt, 2003)
159 # A = 89 # MPa(-3) s(-1)
160 # m = 3.5 # stress exponent
161 # Qc = 480000. # J/mol
162 # j = 0 #Grain size exponent
163 # C_OH = 100 # H/10^6 Si
164 # r_C_OH = 1.2 # Coefficient on C_OH
165 # Volume = 11e-6 # Specific volume for dislocation of wet olivine
166
167 elif body==3:
168     name = 'Enceladus'
169     eccen = 0.0045
170     g_surf = 0.113 # m/s2 surface gravity
171     M = 5.68e26 # mass of Saturn
172     a = 238020000. # avg orbital radius
173     rho_core = 0 # kg/m3 density of iron core approximation
174     rho_rock = 2701 # kg/m3 density of mantle rock approximation
175     rho_crust = 2530 # kg/m3 density of crustal rock approximation
176     rho_hpice = 0
177     rho_water = 1110 #kg/m3 density of liquid water
178     rho_out = 925 #kg/m3 density of ice Ih outer shell
179     R = 252300. # m radius of moon
180     Docean = 13000. # m thickness of liquid ocean
181     Dice = 50000. # m thickness of ice shell
182     Dhpipe = 0 # m thickness of high pressure ice
183     Dcrust = 0 # m thickness of lower density crust
184     Riron = 0 # m radius of iron core
185     mu_f = 0.15 # friction coeff (Chrysotile, Tesei et al., 2018)
186     q = 0.01 # K/m
187     strain = 10 ** (-20) # s-1
188     graph_depth = 250 # Depth shown on plots 3-6
189
190     # #Wet Olivine (Hirst and Kohlstedt, 2003)
191     A=89 #MPa(-3) s(-1)
192     m=3.5 #stress exponent
193     Qc=480000. #J/mol
194     j = 0 # Grain size exponent
195     C_OH = 100 #H/10^6 Si, H contentration (Bai and Kohlstedt, 1992)
196     r_C_OH = 1.2 # Coefficient on C_OH

```

```

196     Volume = 11e-6 # Specific volume for dislocation of wet olivine
197
198 elif body==4:
199     name = 'Moon'
200     eccen = 0.0549
201     g_surf = 1.62 # m/s^2 surface gravity
202     M = 5.972e24 # mass of Earth
203     a = 385000000. # avg orbital radius
204     rho_core = 8000 # kg/m^3 density of iron core approximation
205     rho_rock = 3427 # kg/m^3 density of mantle rock approximation
206     rho_crust = 2530 # kg/m^3 density of crustal rock approximation
207     rho_hpice = 0
208     rho_water = 1025 #kg/m^3 density of liquid water
209     rho_out = 925 #kg/m^3 density of ice Ih outer shell
210     R = 1740000. # m radius of moon
211     Docean = 0. # m thickness of liquid ocean
212     Dice = 0. # m thickness of ice shell
213     Dhpipe = 0 # m thickness of high pressure ice
214     Dcrust = 0 # m thickness of lower density crust
215     Riron = 350000 # m radius of iron core
216     #mu_f = 0.06 # friction coeff (.6 or .3)
217     mu_f = 0.15 # friction coeff (Chrysotile, Tesei et al., 2018)
218     q = 0.002 # K/m
219     strain = 10 ** (-18) # s^-1
220     graph_depth = 500 # Depth shown on plots 3-6
221
222     # #Dry Olivine (Hirth and Kohlstedt, 2003)
223     A = 110000 # MPa^(-3) s^(-1)
224     m = 3.5 # stress exponent
225     Qc = 530000. # J/mol
226     C_OH = 1 #H/10^6 Si, H contentration
227     r_C_OH = 1 # Coefficient on C_OH
228     Volume = 14e-6 # Specific volume for dislocation of dry olivine (Kurato and Rubie,
1997)
229
230
231 #Constants and Parameters
232 G = 6.67408*(10.**(-11.)) #m^3 / kg s^s
233 Rc = R-Dice-Docean-Dhpice # m outer radius of the rocky interior
234 Rm = Rc-Dcrust # m outer radius of mantle
235 Rhp= R-Dice-Docean #m outer radius of high pressure ice
236 Rocean = R-Dice # m outer radius of the liquid ocean
237
238
239 #Uncomment to confirm radius of mantle and core
240 # print(Rm/1000, ' km, Rm')
241 # print(Rc/1000, ' km, Rc')
242
243 #Initialize radius array
244 r=n.zeros((int(R)+1))
245 for i in range(len(r)):
246     r[i]=i
247 dr=r[1]-r[0]
248
249
250
251
252 ### GRAVITY SECTION
253 # Analytical Calculation of Gravity using Enclosed Mass
254 g=n.zeros(len(r))
255 M_encl=0.0
256 M_iron=(4./3.)*n.pi*rho_core*(Riron**3.)
257 M_silicate=(4./3.)*n.pi*rho_rock*((Rc**3.)-(Riron**3.))
258 M_crust=(4./3.)*n.pi*rho_crust*((Rc**3.)-(Rm**3.))
259 M_hp=(4./3.)*n.pi*rho_hpice*((Rhp**3.)-(Rc**3.))
260 M_ocean=(4./3.)*n.pi*rho_water*((Rocean**3.)-(Rhp**3.))

```

```

261 M_ice=(4./3.)*n.pi*rho_out*((R**3.)-(Rocean**3.))
262
263 # Recalculate the density of the mantle based upon the set thickness of the crust
264 rho_rock = (M_silicate - M_crust)/((4./3.)*n.pi*((Rm**3.)-(Riron**3.)))
265 M_rock=(4./3.)*n.pi*rho_rock*((Rm**3.)-(Riron**3.))
266 M_rock_tot = M_rock+M_crust
267
268 ### UNCOMMENT TO CONFIRM THAT NEW MANTLE DENSITY RESULTS IN IDENTICAL TOTAL MASS
269 # print(M_rock_tot, 'kg, mass new method')
270 # print(rho_rock, 'kg/m^3, density new method')
271 # print(M_silicate, 'kg, mass old method')
272 # print(rho_rock, 'kg/m^3, density old method')
273
274
275 for i in range(len(r)):
276     if r[i] <= Riron and r[i] > 0.0:
277         g[i]=(4./3.)*G*n.pi*rho_core*r[i]
278     elif r[i] >Riron and r[i]<=Rm:
279         M_encl=M_iron+((4./3.)*n.pi*rho_rock*((r[i])**3.)-(Riron**3.)))
280         g[i]=(G*M_encl)/((r[i])**2.)
281     elif r[i] > Rm and r[i] <= Rc:
282         M_encl = M_iron+M_rock+((4./3.)*n.pi*rho_crust*((r[i])**3.)-(Rm**3.)))
283         g[i] = (G * M_encl) / ((r[i]) ** 2.)
284     elif r[i] > Rc and r[i] <= Rhp:
285         M_encl=M_iron+M_rock+M_crust+((4./3.)*n.pi*rho_hpice*((r[i])**3.)-(Rc**3.)))
286         g[i]=(G*M_encl)/((r[i])**2.)
287     elif r[i] > Rhp and r[i] <= Rocean:
288         M_encl=M_iron+M_rock+M_crust+M_hp+((4./3.)*n.pi*rho_water*((r[i])**3.)-(Rhp**3.
289         )))
290         g[i]=(G*M_encl)/((r[i])**2.)
291     elif r[i]>Rocean and r[i]<=R:
292         M_encl=M_iron+M_rock+M_crust+M_hp+M_ocean+((4./3.)*n.pi*rho_out*((r[i])**3.)-(
293         Rocean**3.)))
294         g[i]=(G*M_encl)/((r[i])**2.)
295     elif r[i]>R:
296         M_encl=M_iron+M_rock+M_crust+M_hp+M_ocean+M_ice
297         g[i]=(G*M_encl)/((r[i])**2.)
298
299 #Define density as a function of radius
300 def rho(r):
301     if r <= Riron and r > 0.0:
302         rho= rho_core
303     elif r>Riron and r<=Rm:
304         rho= rho_rock
305     elif r>Rm and r<=Rc:
306         rho= rho_crust
307     elif r > Rc and r <= Rhp:
308         rho= rho_hpice
309     elif r > Rhp and r <= Rocean:
310         rho= rho_water
311     elif r>Rocean and r<=R:
312         rho= rho_out
313     else:
314         rho=0.0
315     return rho
316
317 density=n.zeros(len(r))
318 for i in range(len(r)):
319     density[i] = rho(r[i])
320
321 #Print the calculated surface gravity
322 print("{:.3f}".format(g[(len(r)-1)]), '(m/s^2): Surface Gravity')
323
324

```

```

325     ### UNCOMMENT THIS SECTION FOR A PLOT OF GRAVITY VS DEPTH
326     # p.plot(g, (R-r)/1000., "b")
327     # p.title("Gravity vs Radius for " + name)
328     # p.ylabel('Depth (km)')
329     # p.xlabel("Gravity (m/s^2)")
330     # p.show()
331
332
333     ### PRESSURE SECTION
334     dp = density*g*dr
335     P = dp[:, :-1].cumsum()[:, :-1]
336
337     #Print the pressure at the center of the body
338     print(round(P[1]/1000000), '(MPa): Pressure at Center of Body')
339
340
341     #Isolating Properties of Rocky Core Only
342     rcore=n.zeros(int((Rc)+1))
343     gravcore=n.zeros(int((Rc)+1))
344     Pcore=n.zeros(int((Rc)+1))
345     hydrostat=n.zeros(int((Rc)+1))
346     for i in range(len(rcore)):
347         rcore[i]=r[i] #radius in the core
348         gravcore[i]=g[i] #gravity in core
349         Pcore[i]=P[i] #core pressure only
350     drcore=rcore[1]-rcore[0]
351
352     #Print the pressure and gravity at the Seafloor
353     seafloor_grav=gravcore[len(gravcore)-1]
354     seafloor_pressure=Pcore[len(Pcore)-1]
355     print("{:.3f}".format(seafloor_grav), '(m/s^2): SeafloorGravity')
356     print("{:.2f}".format(seafloor_pressure/1000000), '(MPa): Seafloor Pressure' )
357
358
359     if Docean!=0: #For bodies that have an ocean, calculate the hydrostatic pressure gradient
360         #Pore Pressure Calculation
361         hydrostat = ((P[len(Pcore)+1]-P[len(Pcore)])/(r[len(Pcore)+1]-r[len(Pcore)]))*(rcore-
362             r[len(Pcore)])+P[len(Pcore)] #Hydrostatic pressure gradient
363         P_fluid_ratio = 0 #0 for hydrostatic pore fluid pressure, 1 for lithostatic pore
364             fluid pressure
365         Pfluid = hydrostat + (Pcore-hydrostat)*P_fluid_ratio
366         Peffective = Pcore - Pfluid
367         print("{:.3f}".format((Pfluid[len(Pfluid)] - depth)) / 1000000), '(MPa): Fluid
368             pressure at 1000m ')
369     else: #For bodies without an ocean, effective pressure is the same as the lithostatic
370         pressure
371         Peffective = Pcore
372         Pfluid = 0
373
374
375     ### UNCOMMENT THIS SECTION FOR A PLOT OF TOTAL PRESSURE VS DEPTH
376     # p.plot(P/1e6, (R-r)/1000., "k",label='Lithostatic Pressure')
377     # #p.plot(P2/1e6, (R-r)/1000., "r",label='Alternate Pressure')
378     # p.ylabel('Depth (km)')
379     # p.xlabel("Pressure (MPa)")
380     # p.show()
381
382
383     ### UNCOMMENT THIS SECTION FOR PLOTs OF VARIOUS PRESSURES VS DEPTH IN CORE
384     # p.plot(Peffective/(10**6), (R-rcore)/1000., 'g',label='Peffective ')
385     # p.plot(Pfluid/(10**6), (R-rcore)/1000., 'r',label='Hydrostat ')
386     # p.plot(Pcore/(10**6), (R-rcore)/1000., 'c',label='Lithostat ')
387     # p.legend(loc='best')
388     # p.ylabel('Depth (km)')
389     # p.xlabel("Pressure (MPa)")
390     # p.ylim([500,0])

```

```

387 # p.xlim([-200,3000])
388 # p.show()
389
390
391
392 ### FAULT STRENGTH SECTION
393 #Mohr-Coulomb Fault Strength Calculation
394 cohesion = 1.2e6 #Chrysotile (Tesei et al., 2018)
395 faultfriction=(mu_f+n.sqrt(1+(mu_f**2)))**2
396 UCS = 2*cohesion*n.sqrt(faultfriction)
397 print("{:.2f}".format(UCS/1000000), ' (MPa): UCS')
398
399 normal_max = (1. / faultfriction) * (Peffective - UCS) + Pfluid
400 thrust_max=(faultfriction*Peffective) + Pfluid + UCS
401
402 #Differential Strengths
403 diff_normal=(Pcore-normal_max)
404 diff_thrust=-(Pcore-thrust_max)
405
406 #Print out the Fault Strengths at Depth of Choice
407 print("{:.3f}".format((thrust_max[len(thrust_max)-depth])/1000000), ' (MPa): Thrust fault
min strength at ',depth,'m ')
408 print("{:.3f}".format((normal_max[len(normal_max)-depth])/1000000), ' (MPa): Normal fault
min strength at ',depth,'m ')
409 print("{:.3f}".format((diff_normal[len(normal_max)-depth])/1000000), ' (MPa): Normal
fault differential strength at ',depth,'m ')
410 print("{:.3f}".format((diff_thrust[len(thrust_max)-depth])/1000000), ' (MPa): Thrust
fault differential strength at ',depth,'m ')
411 print("{:.3f}".format((Pcore[len(Pcore)-depth])/1000000), ' (MPa): Lithostatic stress at ',
,depth,'m ')
412 print("{:.3f}".format((Peffective[len(Peffective)-depth])/1000000), ' (MPa): Effective
stress at ',depth,'m ')
413
414
415 ### DIURNAL TIDAL STRESS SECTION
416 # Constants and Parameters
417 nu = 0.28 # Unitless, Peridotite (Poisson's Ratio) (Li et al., 2018)
418 SM = 70e9 # Pa, Peridotite (Shear Modulus) (Christensen, 1966)
419
420 if body!=1: #For bodies other than Earth, calculate tidal stress
421
422     m_core=M_iron+M_rock #Mass of silicate core
423     normal=-diff_normal[len(normal_max)-depth] #Pa, Normal fault strength at chosen depth
424
425     Rclkm=Rc-depth #Find the radius of the body at chosen depth below the seafloor
426     sigma=3.*eccen*(M/m_core)*((Rclkm/a)**3)*(5.*rho_rock*seafloor_grav*Rclkm/19.)*2*(1+
nu) #Max stress caused by tidal bulge (3eH) (Collins et al., 2010; Melosh, 1977)
427     reqPfluid = ((faultfriction*Pcore) - Pcore - (faultfriction*sigma) + UCS)/(
faultfriction-1) # Required pore fluid pressure to drive faulting with current
eccentricity
428
429     #Print the Maximum Stress from Tidal Bulge
430     print(math.trunc(round(sigma,-2)), '(Pa): max stress from tidal bulge')
431     print("{:.3f}".format((reqPfluid[len(reqPfluid) - depth]) / 1000000), ' (MPa):
Required pore fluid pressure to drive normal faults at ',depth,'m ')
432     print("{:.3f}".format((Pfluid[len(Pfluid) - depth]) / 1000000), ' (MPa): Hydrostatic
pore fluid pressure at ', depth,'m ')
433     print("{:.2f}".format((reqPfluid[len(reqPfluid) - depth]-Pfluid[len(Pfluid) - depth
])/Pfluid[len(Pfluid) - depth]*100), '% Overpressure to enable normal faulting at ',
depth,'m ')
434
435     #Calculate and Print the Eccentricity Required for Tidal Forces to Fracture Rock
436     eccen_fault=(1./3.)*normal*(m_core/M)*((a/Rclkm)**3)*(19./(5.*rho_rock*seafloor_grav*
Rclkm))/(2*(1+nu))#Invert tidal stress to find eccentricity
437     print("{:.4f}".format(-eccen_fault),'eccentricity req from tides to fracture rock at'
, depth,'m depth')

```

```

438
439
440 ### RADIAL CHANGE AS ROCK FRACTURE MECHANISM
441 #Radial Change required to fracture rock
442 thrust=thrust_max[len(thrust_max)-1000] #Pa, Thrust fault strength at 1 km depth
443 da=(thrust/(2*SM))*((1-nu)/(1+nu))*Rc #Change in radius (Melosh and McKinnon, 1988)
444 print("{:.2f}".format(da), ' (m): Change in radius required to cause thermal fracturing')
445
446 ##Temp change required for radial change to fracture rock
447 a_l = 6.5*10**(-5) #Coefficient of linear thermal expansion
448 delta_t = da/(Rc*a_l) #Change in temperature
449 print("{:.2f}".format(delta_t), ' (K): Change in T required for cooling fracture')
450
451
452 ### CALCULATION OF DUCTILE STRENGTH AND BRITTLE-DUCTILE TRANSITION
453 #Temperature Profiles
454 Tref=274. #K (freezing point of water)
455 T=(-q)*(rcore-Rc)+Tref #K, calculation of temperature profile
456
457 #Ductile Flow Strength
458 Rgas=8.3144 #J/K mol
459 grainsize = 1 # Grain size in microns
460
461 ## Flow law
462 sigma_ductile=((strain/(A*(C_OH**r_C_OH)))*(grainsize**j)*n.exp((Qc+(Pcore*Volume))/(Rgas
463 *T)))**(1/m)
464
465 #intersection of sigma and normal faults
466 bdt = n.argmaxwhere(n.diff(n.sign((thrust_max/(1e6)) - sigma_ductile)) != 0).reshape(-1) + 0
467 #intersection indices of ductile and brittle strength
468 bdtthrust = n.argmaxwhere(n.diff(n.sign((thrust_max/(1e6)) - sigma_ductile)) != 0).reshape(-
469 1) + 0 #intersection indices of ductile and brittle strength
470 bdtnormal = n.argmaxwhere(n.diff(n.sign((normal_max/(1e6)) - sigma_ductile)) != 0).reshape(-
471 1) + 0 #intersection indices of ductile and brittle strength
472 ThrustTransitionPoint=(Rc-rcore[bdtthrust])/1000.
473 NormalTransitionPoint=(Rc-rcore[bdtnormal])/1000.
474 print('Possible Thrust BDT at', ThrustTransitionPoint, 'km in depth') #kilometers depth
475 for BDT
476 print('Possible Normal BDT at', NormalTransitionPoint, 'km in depth') #kilometers depth
477 for BDT
478 #print('Pressure at base of lithosphere:', Pcore[bdt]/1000000, 'MPa')
479 #print(gravcore[bdt], 'gravity at BDT')
480 #print(thrust_max[bdt]/1000000, ' (MPa): Thrust differential strength at base of
481 lithosphere')
482 #print(normal_max[bdt]/1000000, ' (MPa): Normal differential strength at base of
483 lithosphere')
484 #print(lithostat[bdt]/1000000, ' (MPa): BDT lithostat ')
485 #print(hydrostat[bdt]/1000000, ' (MPa): BDT hydrostat ')
486
487
488
489
490 ### GRAPHING SECTION
491 #Graphing code (use radius on the y axis)
492 p.close()
493 if graphoutput==1:
494     p.plot(g,r/1000., "b")
495     p.title("Gravity vs Radius for " + name)
496     p.ylabel('Radius (km) ')
497     p.xlabel("Gravity (m/s^2) ")
498 elif graphoutput==2:
499     p.plot(P/(10**9),r/1000., "b")
500     p.title("Pressure vs Radius for " + name)
501     p.ylabel('Radius (km) ')
502     p.xlabel("Pressure (GPa) ")
503 elif graphoutput==3:

```

```

496     if body!=6: #Do not plot a hydrostat for the moon
497         p.plot(hydrostat/(10**6), (R-rcore)/1000., 'b--', label='Hydrostat')
498     p.plot(Pcore/(10**6), (R-rcore)/1000., 'r', label='Lithostat')
499     p.plot(normal_max/(10**6), (R-rcore)/1000., 'c', label='Normal')
500     p.plot(thrust_max/(10**6), (R-rcore)/1000., 'g', label='Thrust')
501     p.ylim([graph_depth, 0])
502     p.xlim([0, 2000])
503     p.title('Fault Strength for ' + name)
504     p.ylabel('Depth (km)')
505     p.xlabel('Pressure (MPa)')
506 elif graphoutput==4:
507     if body!=4: #Do not plot a hydrostat for the moon
508         p.plot(hydrostat / (10 ** 6), (R - rcore) / 1000., 'b--', label='Hydrostat')
509     p.plot(Pcore / (10 ** 6), (R - rcore) / 1000., 'r', label='Lithostat')
510     p.plot(normal_max / (10 ** 6), (R - rcore) / 1000., 'c', label='Normal')
511     p.plot(thrust_max / (10 ** 6), (R - rcore) / 1000., 'g', label='Thrust')
512     p.plot(sigma_ductile, (R - rcore)/1000., "c:", label='Ductile Low Strain')
513     p.xscale('log')
514     p.ylabel('Depth (km)')
515     p.xlim([1e-6, 1e8])
516     p.ylim([graph_depth, 0])
517     p.title('Log Plot Fault Strength for ' + name)
518     p.xlabel('Pressure (MPa)')
519 elif graphoutput==5:
520     p.plot(T, (R - rcore)/1000., 'r', label='Temperature')
521     p.title('Geotherm for ' + name)
522     p.ylim([graph_depth, 0])
523     p.xlabel('Temperature (K)')
524     p.ylabel('Depth (km)')
525 elif graphoutput==6:
526     #p.plot(hydrostat/(10**6), (Rc-rcore)/1000., 'b--', label='Hydrostat')
527     #p.plot(lithostat/(10**6), (R-rcore)/1000., 'r', label='Lithostat')
528     p.plot(-diff_normal/(10**6), (R-rcore)/1000., 'y', label='Normal')
529     #p.plot(normal_max/(10**6), rcore/1000., 'c', label='Normal 40%')
530     p.plot(diff_thrust/(10**6), (R-rcore)/1000., 'm', label='Thrust')
531     #p.plot(thrust_max/(10**6), rcore/1000., 'g', label='Thrust 40%')
532     p.plot(sigma_ductile, (R-rcore)/1000., "c:", label='Wet Olivine Dislocation')
533     p.plot(-sigma_ductile, (R-rcore)/1000., "c:")
534     p.ylim([graph_depth, 0])
535     p.xlim([-250, 250]) # For Europa
536     #p.xlim([-13000, 13000]) # For Earth
537     p.ylabel('Depth (km)')
538     p.title('Plot Fault Strength for ' + name)
539     p.xlabel('Pressure (MPa)')
540 elif graphoutput == 7:
541     # p.plot(hydrostat/(10**6), (Rc-rcore)/1000., 'b--', label='Hydrostat')
542     # p.plot(lithostat/(10**6), (R-rcore)/1000., 'r', label='Lithostat')
543     p.plot(normal_max / (10 ** 6), (R - rcore) / 1000., 'y', label='Normal')
544     p.plot(thrust_max / (10 ** 6), (R - rcore) / 1000., 'm', label='Thrust')
545     p.plot(sigma_ductile, (R - rcore) / 1000., "c:", label='Dislocation')
546     p.plot(-sigma_ductile, (R - rcore) / 1000., "c:")
547     p.ylim([graph_depth, 0])
548     p.xlim([-1300, 1300]) # For Europa
549     # p.xlim([-13000, 13000]) # For Earth
550     p.ylabel('Depth (km)')
551     p.title('Plot Fault Strength for ' + name)
552     p.xlabel('Pressure (MPa)')
553 elif graphoutput==8:
554     exit()
555 p.legend(loc='best')
556 p.show()
557
558
559

```