

Investigation of Material Properties at Extreme Conditions Through the Virtual Laboratory of Earth and Planetary Materials (VLab)

Caroline Qian, Renata Wentzcovitch, and Gaurav Shukla

The purpose of this project is to improve on the existing code in VLab for calculating the thermodynamic properties of minerals at extreme temperatures and pressures. The resulting program is able to read tables of various properties generated by VLab and use those to calculate a new property, the gradient. This new data can then be graphed, using integration and interpolation methods to obtain points that may not already be in the table, and used as a model of the geothermal gradient of the Earth. Specifically, our goal is to obtain the temperature versus pressure gradient in the Earth's lower mantle by integrating the adiabatic gradient using the boundary condition of pressure being 23 GPa at 1900 K. This program currently calculates the properties of different aggregates of two materials, but the generalization to multi-phase aggregates is trivial, except for the coding part, which must also be generalized. The two-phase version presented here works perfectly.

The gradient is calculated as the change in temperature over the change in pressure at constant entropy, and can also be calculated as an expression of several properties:

$$G = \left. \frac{dT}{dP} \right|_s = \frac{\alpha VT}{C_p}$$

Where α is thermal expansivity in $10^{-5}K^{-1}$, V is volume in \AA^3 per unit cell (four molecules per unit cell for MgSiO_3 and MgO), T is the temperature in Kelvin, and the specific heat capacity C_p is in $\text{Jmol}^{-1}K^{-1}$. Because of these units, we will need to multiply the value of G by 1.0×10^{-26} in order to obtain the value in units of K/GPa .

VLab already produces data tables for α , V , and C_p for various minerals. For the examples presented in this report, data for MgSiO_3 and MgO are used, which are interpreted as species 1 and 2, respectively. Once the user has downloaded these tables from the VLab website, he can run the code, which will read the tables and produce a new one labeled "adiabat.txt" that contains all the gradient values at the corresponding temperatures and pressures of the input tables.

The second part of the program generates a geothermal gradient depending on the starting temperature and pressure the user inputs. This starting temperature and pressure are stored and the "interp" function of the program determines whether the corresponding gradient value is in the table or must be interpolated using the "polate" function.

The "polate" function (page 12) is the bilinear interpolation method, where the "x_one" and "x_two" inputs are the pressure values determined by the "interp" function and "y_one" and "y_two" are the temperatures. The "interp" function (page 10) determines these values depending on whether the temperature and/or pressure is/are at a maximum of the table.

Finally, the "rungekutta_T" function (page 12) is used to return the next temperature and pressure the gradient will be evaluated at. This function makes use of the Runge-Kutta integration method, which produces a result based on the weighted average of four calculated increments calculated with the step size. For this example, a step size of 3 is used because the change in pressures of the input tables is 3 GPa.

The process then repeats until the program reaches the maximum temperature or pressure provided by the input tables. The values stored in the “graphTPG” array can then be graphed to show the results of the geothermal gradient line compared to the entire gradient table calculated earlier and stored in “adiabat.txt”.

The following figures are the results of graphing the “graphTPG” table in MatLab (the blue line is geothermal gradient; the surface is the gradient values from “adiabat.txt”):

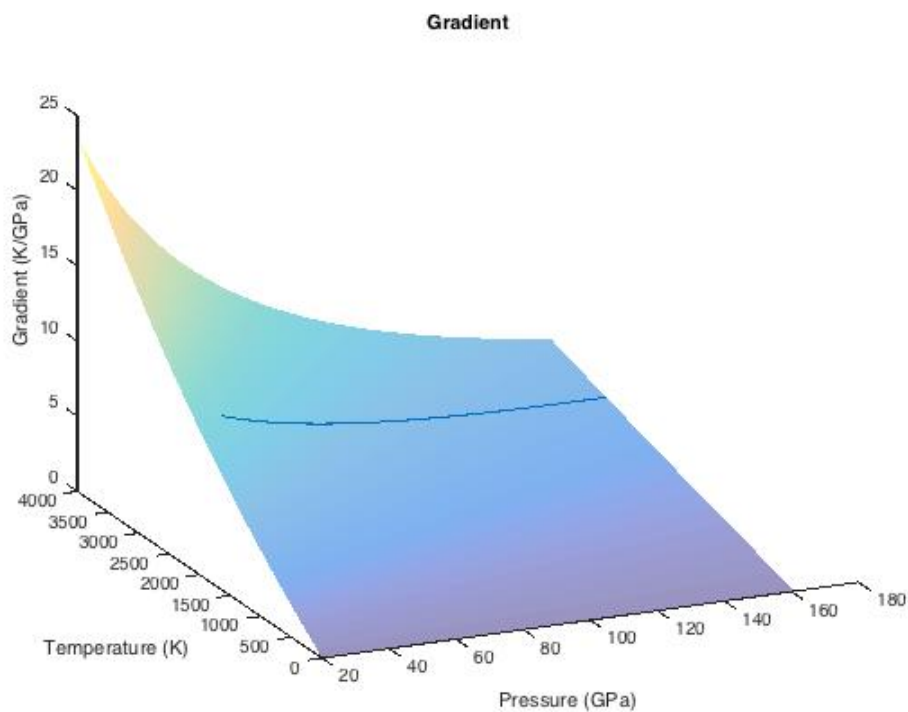
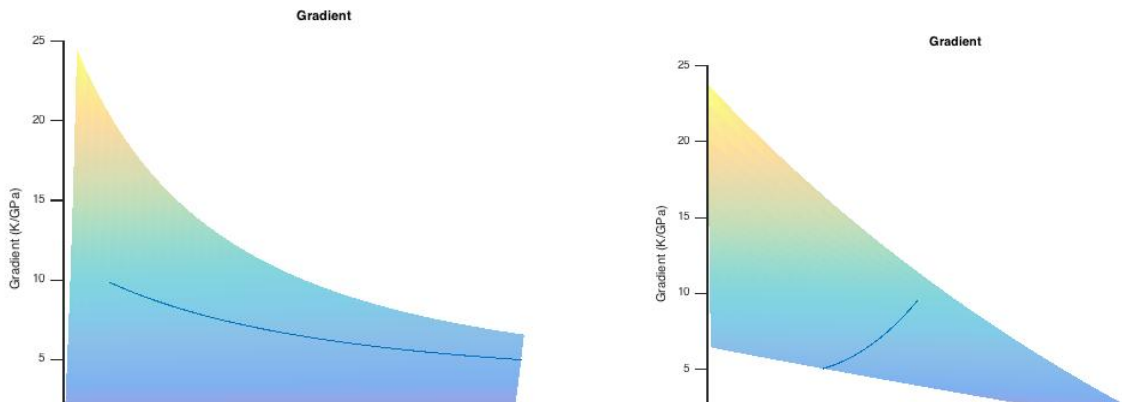


Figure 1



```
1 program gradient
2 !this program reads in user inputs of the downloaded files from the Vlab website,
3 !assuming that the file names are alpha.dat, Cp.dat, and vol.dat
4 ! the first column of each matrix is the temperature
5 ! R. Wentzcovitch, C. Qian, G. Shukla, 2013 - 2014
6
7 integer eof, n, skip, count_lines, k, c, num_species
8 !depending on the number of species, create the corresponding number of arrays for
9 !alpha, cp, vol, and mv (molar volume)
10 real, allocatable, dimension (:, :) :: alpha_1, alpha_2, cp_1, cp_2, vol_1, vol_2, &
11 mv_1, mv_2
12 real, allocatable, dimension (:, :) :: per_unit, adiabat, alpha_agg, cp_agg, &
13 vol_agg, graphTPG
14 !h is the step size for runge kutta, to be input by the user. n_1 and n_2 are the
15 !mole fraction of species 1 and 2 in the aggregate
16 real dummy, T, P, h, n_1, n_2
17 n = 1
18 num_species = 2 !change according to the number of species in the aggregate
19 k = 49 !change k appropriately and replace 49 in the format 110 and 130 lines.
20 skip = 0
21 c = 0 !variable to count the number of rows in an input table
22 n_1 = 0.8
23 n_2 = 0.2
24 allocate(per_unit(num_species, 1))
25
26
27 110 format (f15.6, 6x, f10.6, 49(6x, f10.6))
28 120 format (f10.6, 6x)
29 130 format (23x, f10.6, 49(6x, f10.6))
30 140 format (21x, f10.6, 49(6x, f10.6))
31
32 !ALL DATA TABLES SHOULD BE THE SAME DIMENSIONS, WITH THE SAME STEP SIZES FOR PRESSURES
33 !AND TEMPERATURES (e.g. change in pressure is 3 GPa and change in temperature is 5K)
34
35 do i = 1, num_species
36     print *, 'Enter number of molecules per unit cell for species', i
37     read (*,*) per_unit(i, 1)
38 end do
39
40 print *, 'Enter a Temperature (K), Pressure (GPa), and step size (for Runge-Kutta method):'
41 read (*,*) T, P, h
42
43 !This segment determines how many rows will be in the data tables. It is assumed that
44 !alpha.dat, Cp.dat, and vol.dat will have the same number of rows.
45 open (unit = 31, file = 'alpha_1.dat', status = 'old')
46 read (unit = 31, fmt = 120, iostat = eof) dummy
47 70 if (eof .ge. 0 .and. skip .le. 0) then
48     skip = skip + 1
49     read (unit = 31, fmt = 120, iostat = eof) dummy
50     go to 70
51
52     else if (eof .ge. 0) then
53         c = c + 1
54         read (unit = 31, fmt = 120, iostat = eof) dummy
55         go to 70
56     end if
57 close (31)
58
59 !create matrixes for alpha, Cp, vol, and the adiabat with the appropriate number of rows
60 !and columns.
61 !If more than two species, create appropriate arrays for those as well.
62
63 allocate(alpha_1(c, k))
64 allocate (cp_1(c, k))
65 allocate (vol_1(c, k))
66 allocate(mv_1(c, k))
67
68 allocate(alpha_2(c, k))
69 allocate (cp_2(c, k))
70 allocate (vol_2(c, k))
71 allocate(mv_2(c, k))
72
73 allocate (alpha_agg(c, k))
74 allocate (cp_agg(c, k))
75 allocate (vol_agg(c, k))
76
77 allocate(adiabat(c, k))
78 allocate(graphTPG(c, 3))
79
80 !For this version of the program, we assume that the number of species is 2.
81 !Depending on the value of num_species, copy and paste the code between the *s
82 !(lines 85 through 166) and change the unit numbers and file names accordingly.
83 !For simplicity, files should be named alpha_1.dat, alpha_2.dat, etc.
84
85 !*
86
87 !read alpha_1
88 n = 1
89 skip = 0
90 !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
91 alpha_1 (1,1) = 0
92
93 open (unit = 31, file = 'alpha_1.dat', status = 'old')
94 read (unit = 31, fmt = 130, iostat = eof) alpha_1(n, 2:k)
95 10 if (eof .ge. 0 .and. skip .le. 0) then
96     skip = skip + 1
```

```
97     read (unit = 31, fmt = 130, iostat = eof) alpha_1(n, 2:k)
98     go to 10
99     else if (eof .ge. 0) then
100         n = n + 1
101         read (unit = 31, fmt = 110, iostat = eof) alpha_1(n, 1:k)
102         go to 10
103     end if
104 close (31)
105
106 !if the file hasn't been created before, status will have to be changed to 'new'
107 open (unit = 35, file = 'alphanew_1.txt', status = 'old')
108
109 do i=1, c
110     write (35, *) alpha_1 (i, 1:k)
111 enddo
112 close (35)
113
114 !read Cp_1
115 n = 1
116 skip = 0
117 !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
118 cp_1 (1,1) = 0
119
120 open (unit = 32, file = 'Cp_1.dat', status = 'old')
121 read (unit = 32, fmt = 130, iostat = eof) cp_1(n, 2:k)
122 20 if (eof .ge. 0 .and. skip .le. 0) then
123     skip = skip + 1
124     read (unit = 32, fmt = 130, iostat = eof) cp_1(n, 2:k)
125     go to 20
126 else if (eof .ge. 0) then
127     n = n + 1
128     read (unit = 32, fmt = 110, iostat = eof) cp_1(n, 1:k)
129     go to 20
130 end if
131 close (32)
132
133 !if the file hasn't been created before, status will have to be changed to 'new'
134 open (unit = 36, file = 'cpnew_1.txt', status = 'old')
135 do i=1, c
136     write (36, *) cp_1(i, 1:k)
137 enddo
138 close (36)
139
140 !read vol_1
141 n = 1
142 skip = 0
143 !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
144 vol_1 (1,1) = 0
145
146 open (unit = 33, file = 'vol_1.dat', status = 'old')
147 read (unit = 33, fmt = 130, iostat = eof) vol_1(n, 2:k)
148 30 if (eof .ge. 0 .and. skip .le. 0) then
149     skip = skip + 1
150     read (unit = 33, fmt = 130, iostat = eof) vol_1(n, 2:k)
151     go to 30
152 else if (eof .ge. 0) then
153     n = n + 1
154     read (unit = 33, fmt = 110, iostat = eof) vol_1(n, 1:k)
155     go to 30
156 end if
157 close (33)
158
159 !if the file hasn't been created before, status will have to be changed to 'new'
160 open (unit = 37, file = 'volnew_1.txt', status = 'old')
161 do i=1, c
162     write (37, *) vol_1 (i, 1:k)
163 enddo
164 close (37)
165
166 !*
167
168 !read alpha_2
169 n = 1
170 skip = 0
171 !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
172 alpha_2 (1,1) = 0
173
174 open (unit = 41, file = 'alpha_2.dat', status = 'old')
175 read (unit = 41, fmt = 130, iostat = eof) alpha_2(n, 2:k)
176 40 if (eof .ge. 0 .and. skip .le. 0) then
177     skip = skip + 1
178     read (unit = 41, fmt = 140, iostat = eof) alpha_2(n, 2:k)
179     go to 40
180 else if (eof .ge. 0) then
181     n = n + 1
182     read (unit = 41, fmt = 110, iostat = eof) alpha_2(n, 1:k)
183     go to 40
184 end if
185 close (41)
186
187 !if the file hasn't been created before, status will have to be changed to 'new'
188 open (unit = 45, file = 'alphanew_2.txt', status = 'old')
189
190 do i=1, c
191     write (45, *) alpha_2 (i, 1:k)
192 enddo
```

```
193     close (45)
194
195 !read Cp_2
196     n = 1
197     skip = 0
198     !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
199     cp_2 (1,1) = 0
200
201     open (unit = 42, file = 'Cp_2.dat', status = 'old')
202     read (unit = 42, fmt = 130, iostat = eof) cp_2(n, 2:k)
203 50    if (eof .ge. 0 .and. skip .le. 0) then
204         skip = skip + 1
205         read (unit = 42, fmt = 140, iostat = eof) cp_2(n, 2:k)
206         go to 50
207     else if (eof .ge. 0) then
208         n = n + 1
209         read (unit = 42, fmt = 110, iostat = eof) cp_2(n, 1:k)
210         go to 50
211     end if
212     close (42)
213
214     !if the file hasn't been created before, status will have to be changed to 'new'
215     open (unit = 46, file = 'cpnew_2.txt', status = 'old')
216     do i=1, c
217         write (46, *) cp_2(i, 1:k)
218     enddo
219     close (46)
220
221 !read vol_2
222     n = 1
223     skip = 0
224     !this is for table formatting purposes: there is no T/P entry for (1,1) in the array
225     vol_2 (1,1) = 0
226
227     open (unit = 43, file = 'vol_2.dat', status = 'old')
228     read (unit = 43, fmt = 130, iostat = eof) vol_2(n, 2:k)
229 60    if (eof .ge. 0 .and. skip .le. 0) then
230         skip = skip + 1
231         read (unit = 43, fmt = 140, iostat = eof) vol_2(n, 2:k)
232         go to 60
233     else if (eof .ge. 0) then
234         n = n + 1
235         read (unit = 43, fmt = 110, iostat = eof) vol_2(n, 1:k)
236         go to 60
237     end if
238     close (43)
239
240     !if the file hasn't been created before, status will have to be changed to 'new'
241     open (unit = 47, file = 'volnew_2.txt', status = 'old')
242     do i=1, c
243         write (47, *) vol_2 (i, 1:k)
244     enddo
245     close (47)
246
247 !calculate molar volume of aggregate (vol_agg)
248
249 do i = 1, c
250     do j = 1, k
251         if (j .le. 1 .or. i .le. 1) then
252             mv_1 (i, j) = vol_1(i, j)
253         else if (j .ge. 2) then
254             mv_1(i, j) = 6.0221412927E23 * vol_1(i, j)/per_unit(1, 1)
255         end if
256     enddo
257 enddo
258
259 do i = 1, c
260     do j = 1, k
261         if (j .le. 1 .or. i .le. 1) then
262             mv_2 (i, j) = vol_2(i, j)
263         else if (j .ge. 2) then
264             mv_2(i, j) = 6.0221412927E23 * vol_2(i, j)/per_unit(2, 1)
265         end if
266     enddo
267 enddo
268     open (unit = 51, file = 'vol_agg.txt', status = 'old')
269 do i = 1, c
270     do j = 1, k
271         if (j .le. 1 .or. i .le. 1) then
272             vol_agg (i, j) = vol_1(i, j)
273         else if (j .ge. 2) then
274             vol_agg(i, j) = n_1*mv_1(i, j) + n_2*mv_2(i, j)
275         end if
276     enddo
277     write (51, *) vol_agg(i, 1:k)
278 enddo
279
280 !calculate alpha of aggregate (alpha_agg)
281 open (unit = 53, file = 'alpha_agg.txt', status = 'old')
282 do i = 1, c
283     do j = 1, k
284         if (j .le. 1 .or. i .le. 1) then
285             alpha_agg (i, j) = alpha_1(i, j)
286         else if (j .ge. 2) then
287             alpha_agg(i, j) = (n_1*mv_1(i, j)*alpha_1(i, j))/vol_agg(i, j) + &
288                 (n_2*mv_2(i, j)*alpha_2(i, j))/vol_agg(i, j)

```

```
289     end if
290   enddo
291   write (53, *) alpha_agg(i,1:k)
292 enddo
293
294
295
296 !calculate Cp of aggregate (cp_agg)
297 open (unit = 52, file = 'cp_agg.txt', status = 'new')
298 do i = 1, c
299   do j = 1, k
300     if (j .le. 1 .or. i .le. 1) then
301       cp_agg (i, j) = cp_1(i, j)
302     else if (j .ge. 2) then
303       cp_agg(i, j) = n_1*cp_1(i, j) + n_2*cp_2(i, j)
304     end if
305   enddo
306   write (52, *) cp_agg(i,1:k)
307 enddo
308
309 !calculate adiabatic gradient of aggregate
310
311 !if the file hasn't been created before, status will have to be changed to 'new'
312 open (unit = 44, file = 'adiabat.txt', status = 'old')
313
314 do i = 1, c
315   do j = 1, k
316     if (j .le. 1 .or. i .le. 1) then
317       adiabat (i, j) = alpha_1(i, j)
318     else if (j .ge. 2) then
319       adiabat(i, j) = 1.0E-26* alpha_agg(i, j) * vol_agg(i, j) * &
320         alpha_1(i, 1)/cp_agg(i, j)
321     end if
322   enddo
323   write (44, *) adiabat(i,1:k) !store adiabatic gradient in file adiabat.txt.
324 enddo
325   close (44)
326
327 !interpolating gradient
328
329 n = 1
330
331   graphTPG(n, 1) = T
332   graphTPG(n, 2) = P
333   graphTPG(n, 3) = interp(T, P)
334
335 n = 2
336
337 do while (n .le. c)
338   graphTPG (n, 2) = graphTPG (n-1, 2) + h
339   if (graphTPG(n, 2) .le. adiabat(1, k)) then
340     !alpha_1(i, 1) = temperature, 1.0E-26 is a factor we must multiply by because of the
341     !units of alpha (10^-5 K^-1), Cp (J/(mol K)), and volume (A^3)
342     graphTPG(n, 1) = rungekutta_T (graphTPG(n-1, 1), graphTPG(n-1, 2), h)
343     graphTPG(n, 3) = interp(graphTPG(n, 1), graphTPG(n, 2))
344     n = n+1
345   else if (n .gt. c .or. graphTPG(n, 2) .gt. adiabat(1, k)) then
346     exit
347   end if
348 end do
349
350 !if the file hasn't been created before, status will have to be changed to 'new'
351 open (unit = 38, file = 'gradient.txt', status = 'old')
352 do i = 1, c
353   write(38, *) graphTPG(i, 1:3)
354 enddo
355   close(38)
356
357
358 contains
359
360 function interp (T, P)
361   real T, P, interp
362   integer x, y, z, w
363   x = 1
364   y = 1
365   z = 1
366   w = 1
367
368   do i = 1, c
369     if (adiabat(i, 1) .le. T) then
370       x = i
371     end if
372   end do
373
374   do j = 1, k
375     if (adiabat(1, j) .le. P) then
376       y = j
377     end if
378   end do
379
380   if (adiabat(1, y) .eq. P .and. adiabat(x, 1) .eq. T) then
381     interp = adiabat (x, y)
382     end if
383
384   if (x .eq. c .and. y .lt. k) then
```

```
385     z = x - 1
386     w = y + 1
387     end if
388
389     if (x .lt. c .and. y .eq. k) then
390         z = x + 1
391         w = y - 1
392     end if
393
394     if (x .eq. c .and. y .eq. k) then
395         z = x - 1
396         w = y - 1
397     end if
398
399     if (x .lt. c .and. y .lt. k) then
400         z = x + 1
401         w = y + 1
402     end if
403
404     interp = polate (T, P, adiabat (x, 1), adiabat (z, 1), adiabat(1, y), &
405     adiabat (1, w), adiabat (x, y), adiabat (x, w), adiabat (z, y), adiabat (z, w))
406     return
407 end function interp
408
409 function polate (T, P, x_one, x_two, y_one, y_two, f_11, f_12, f_21, f_22)
410     real f_one, f_two, T, P, f_11, f_12, f_21, f_22, x_one, x_two, y_one, y_two, polate
411
412     f_one = (x_two - T)/(x_two - x_one)*f_11 + (T - x_one)/(x_two - x_one) * f_21
413     f_two = (x_two - T)/(x_two - x_one)*f_12 + (T - x_one)/(x_two - x_one) * f_22
414     polate = (y_two - P)/(y_two - y_one) * f_one + (P - y_one)/(y_two - y_one) * f_two
415     return
416 end function polate
417
418 function rungekutta_T (T, P, h) ! returns T, P increases by h each step
419     real T, P, h, k_1, k_2, k_3, k_4, rungekutta_T
420
421     k_1 = interp (T, P)
422     k_2 = interp (T + 0.5 * h * k_1, P + 0.5 * h)
423     k_3 = interp (T + 0.5 * h * k_2, P + 0.5 * h)
424     k_4 = interp (T + h * k_3, P + h)
425
426     rungekutta_T = T + (h/6) * (k_1 + 2 * k_2 + 2 * k_3 + k_4)
427     return
428 end function rungekutta_T
429
430 end program gradient
```