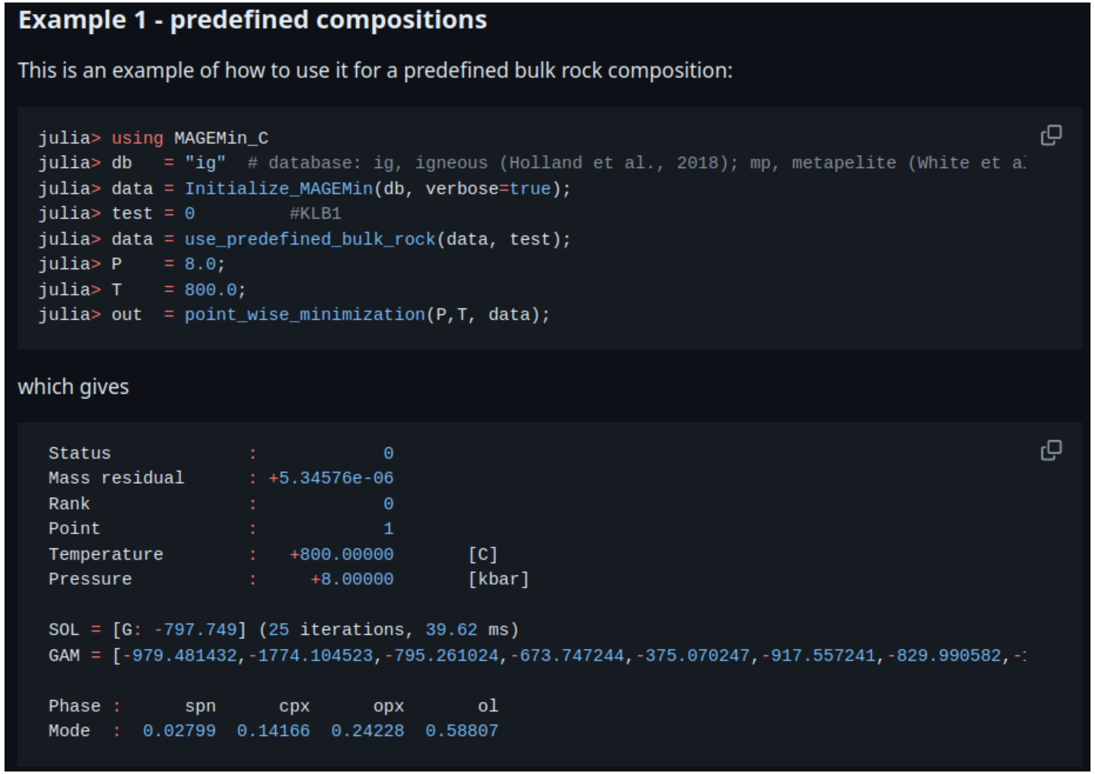
MAGEMin_C, the julia interface: introduction



Example 1 - predefined compositions

This is an example of how to use it for a predefined bulk rock composition:

```
julia> using MAGEMin_C
julia> db   = "ig"  # database: ig, igneous (Holland et al., 2018); mp, metapelite (White et a
julia> data = Initialize_MAGEMin(db, verbose=true);
julia> test = 0         #KLB1
julia> data = use_predefined_bulk_rock(data, test);
julia> P    = 8.0;
julia> T    = 800.0;
julia> out  = point_wise_minimization(P,T, data);
```

which gives

```
 Status              :            0
 Mass residual       : +5.34576e-06
 Rank                :            0
 Point               :            1
 Temperature         :   +800.00000       [C]
 Pressure            :     +8.00000       [kbar]

 SOL = [G: -797.749] (25 iterations, 39.62 ms)
 GAM = [-979.481432,-1774.104523,-795.261024,-673.747244,-375.070247,-917.557241,-829.990582,-

 Phase :      spn       cpx       opx        ol
 Mode  :  0.02799   0.14166   0.24228   0.58807
```

MAGEMin_C, the julia interface: installation

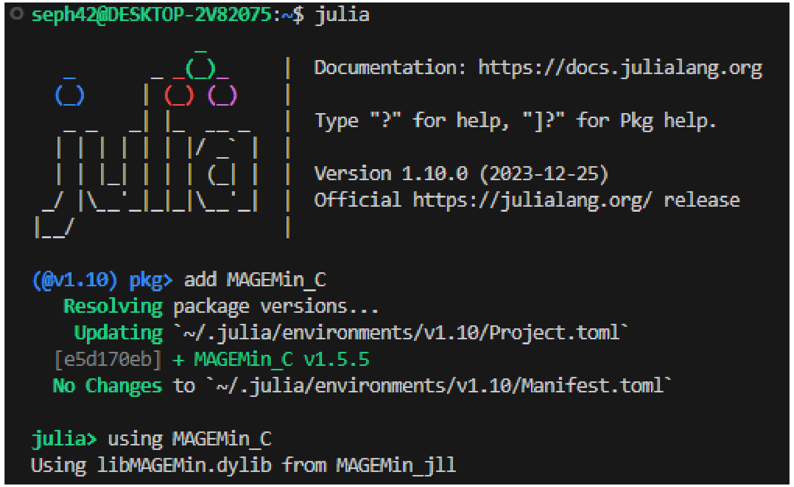
https://github.com/ComputationalThermodynamics/MAGEMin_C.jl

Install MAGEMin_C

```
julia> ]                   # opens the package manager
pkg> add MAGEMin_C         # MAGEMin_C
```

Load MAGEMin_C

```
julia> using MAGEMin_C   # load MAGEMin_C
```

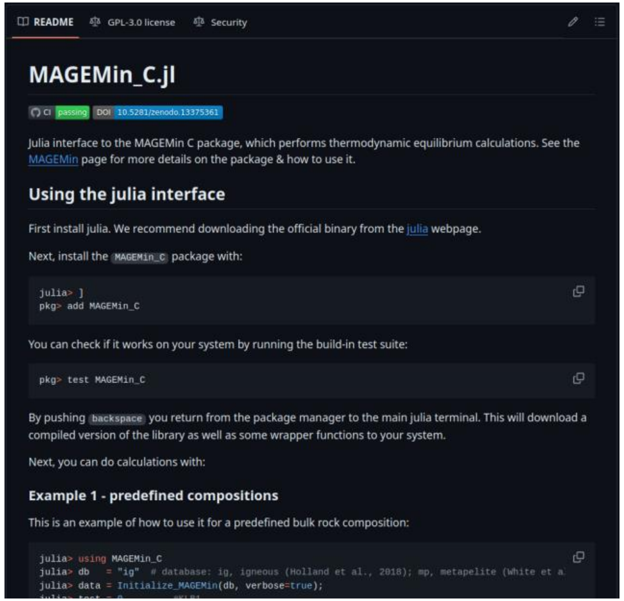


```
seph42@DESKTOP-2V82075:~$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.0 (2023-12-25)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

(@v1.10) pkg> add MAGEMin_C
   Resolving package versions...
    Updating `~/.julia/environments/v1.10/Project.toml`
  [e5d170eb] + MAGEMin_C v1.5.5
  No Changes to `~/.julia/environments/v1.10/Manifest.toml`

julia> using MAGEMin_C
Using libMAGEMin.dylib from MAGEMin_jll
```

MAGEMin_C, the julia interface: online documentation

https://github.com/ComputationalThermodynamics/MAGEMin_C.jl

- Numerous examples outside what will be presented during the short course are given in the README.md
- Simply access the MAGEMin_C.jl github link and scrolldown to see them!



MAGEMin_C.jl

CI passing DOI 10.5281/zenodo.13375361

Julia interface to the MAGEMin C package, which performs thermodynamic equilibrium calculations. See the MAGEMin page for more details on the package & how to use it.

Using the julia interface

First install julia. We recommend downloading the official binary from the julia webpage.

Next, install the `MAGEMin_C` package with:

```
julia> ]
pkg> add MAGEMin_C
```

You can check if it works on your system by running the build-in test suite:

```
pkg> test MAGEMin_C
```

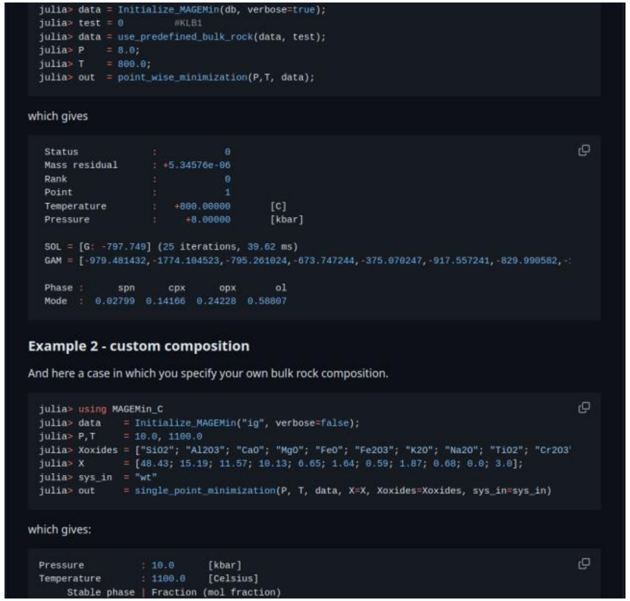
By pushing `backspace` you return from the package manager to the main julia terminal. This will download a compiled version of the library as well as some wrapper functions to your system.

Next, you can do calculations with:

Example 1 - predefined compositions

This is an example of how to use it for a predefined bulk rock composition:

```
julia> using MAGEMin_C
julia> db   = "ig"  # database: ig, igneous (Holland et al., 2018); mp, metapelite (White et a
julia> data = Initialize_MAGEMin(db, verbose=true);
```



```
julia> data = Initialize_MAGEMin(db, verbose=true);
julia> test = 0         #KLB1
julia> data = use_predefined_bulk_rock(data, test);
julia> P    = 8.0;
julia> T    = 800.0;
julia> out  = point_wise_minimization(P,T, data);
```

which gives

```
Status              :            0
Mass residual       : +5.34576e-06
Rank                :            0
Point               :            1
Temperature         :   +800.00000       [C]
Pressure            :     +8.00000       [kbar]

SOL = [G: -797.749] (25 iterations, 39.62 ms)
GAM = [-979.481432,-1774.104523,-795.261024,-673.747244,-375.070247,-917.557241,-829.990582,-

Phase :      spn      cpx      opx       ol
Mode  :  0.02799  0.14166  0.24228  0.58807
```

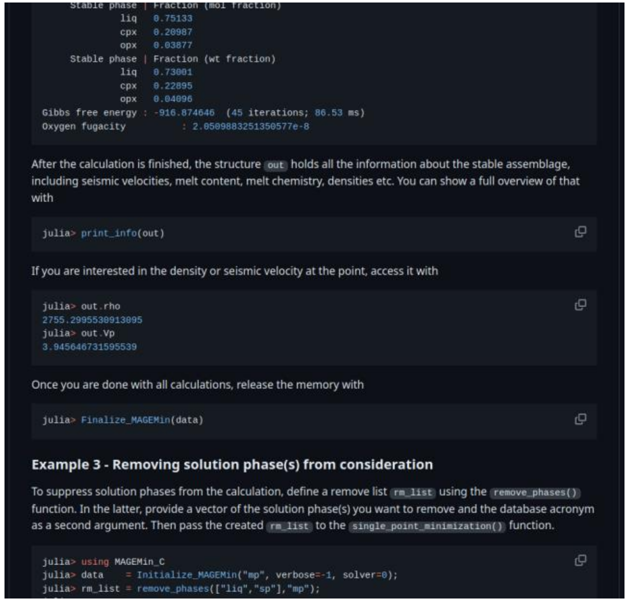
Example 2 - custom composition

And here a case in which you specify your own bulk rock composition.

```
julia> using MAGEMin_C
julia> data    = Initialize_MAGEMin("ig", verbose=false);
julia> P,T     = 10.0, 1100.0
julia> Xoxides = ["SiO2"; "Al2O3"; "CaO"; "MgO"; "FeO"; "Fe2O3"; "K2O"; "Na2O"; "TiO2"; "Cr2O3'
julia> X       = [48.43; 15.19; 11.57; 10.13; 6.65; 1.64; 0.59; 1.87; 0.68; 0.0; 3.0];
julia> sys_in  = "wt"
julia> out     = single_point_minimization(P, T, data, X=X, Xoxides=Xoxides, sys_in=sys_in)
```

which gives:

```
Pressure          : 10.0      [kbar]
Temperature       : 1100.0    [Celsius]
     Stable phase | Fraction (mol fraction)
```



```
     Stable phase | Fraction (mol fraction)
              liq   0.75133
              cpx   0.20987
              opx   0.03877
     Stable phase | Fraction (wt fraction)
              liq   0.73001
              cpx   0.22895
              opx   0.04096
Gibbs free energy : -916.874646  (45 iterations; 86.53 ms)
Oxygen fugacity          : 2.0509883251350577e-8
```

After the calculation is finished, the structure `out` holds all the information about the stable assemblage, including seismic velocities, melt content, melt chemistry, densities etc. You can show a full overview of that with

```
julia> print_info(out)
```

If you are interested in the density or seismic velocity at the point, access it with

```
julia> out.rho
2755.2995530913095
julia> out.Vp
3.945646731595539
```

Once you are done with all calculations, release the memory with

```
julia> Finalize_MAGEMin(data)
```

Example 3 - Removing solution phase(s) from consideration

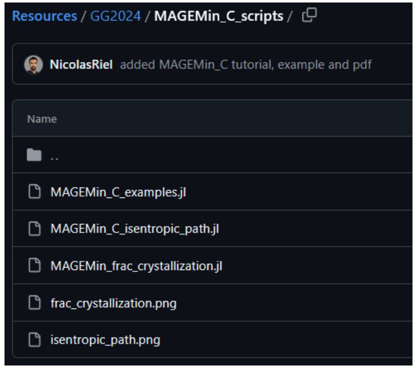
To suppress solution phases from the calculation, define a remove list `rm_list` using the `remove_phases()` function. In the latter, provide a vector of the solution phase(s) you want to remove and the database acronym as a second argument. Then pass the created `rm_list` to the `single_point_minimization()` function.

```
julia> using MAGEMin_C
julia> data    = Initialize_MAGEMin("mp", verbose=-1, solver=0);
julia> rm_list = remove_phases(["liq","sp"],"mp");
```

Moreover, the testing framework can provide more knowledge:
https://github.com/ComputationalThermodynamics/MAGEMin_C.jl/blob/main/test/tests.jl

MAGEMin_C, the julia interface: let's get started!

- MAGEMin_C allow you to write phase equilibrium **Julia** scripts fitted to your needs
- Numerous resources to learn Julia can be found online e.g., https://julialang.org/learning/tutorials/
- Overall, **Julia** programming style is quite like **Matlab** and being a Matlab user will help to write in Julia
- Here we are going to explore the basic functionality **MAGEMin_C** and we will go through one advanced example on how to compute iso-entropic paths (fixed entropy during decompression, applied to MORB genesis)
- Note that if you struggle, the scripts presented in this tutorial are available in the resources, in the GG2024 folder



Resources / GG2024 / MAGEMin_C_scripts /

NicolasRiel added MAGEMin_C tutorial, example and pdf

Name
..
MAGEMin_C_examples.jl
MAGEMin_C_isentropic_path.jl
MAGEMin_frac_crystallization.jl
frac_crystallization.png
isentropic_path.png

MAGEMin_C, simple example

- The following example performs an equilibrium given the igneous database “ig” and the predefined test for KLB-1 peridotite
  Note that: “ig”, igneous; “mp”, metapelite; “mb”, metabasite; “um”, ultramafic

```julia
using MAGEMin_C         # load MAGEMin

# Initialize database
data        =   Initialize_MAGEMin("ig", verbose=true);
test        =   0         #KLB1
data        =   use_predefined_bulk_rock(data, test);

# Call optimization routine for given P & T & bulk_rock
P           =   8.0
T           =   800.0
out         =   single_point_minimization(P,T, data);
Finalize_MAGEMin(data)
```

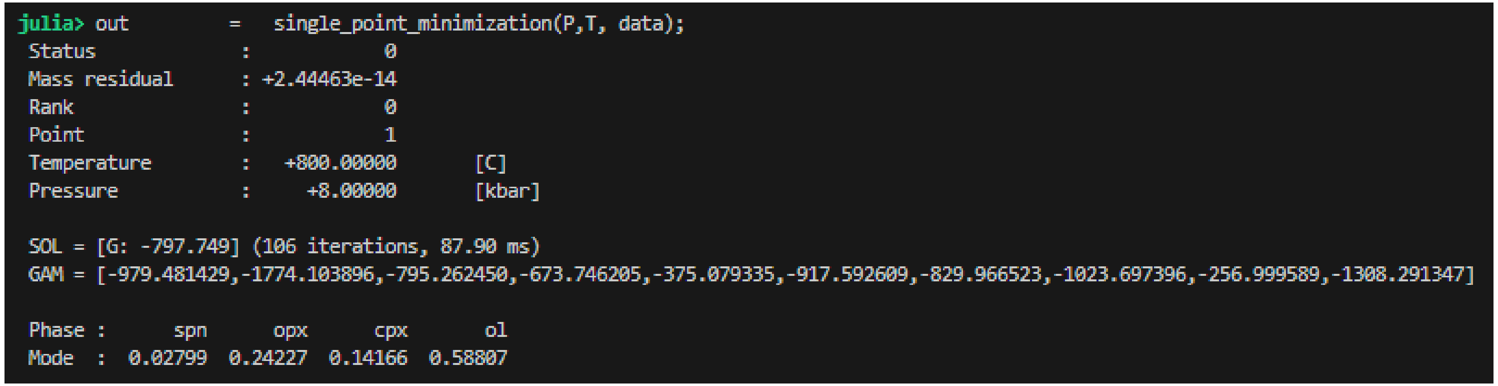
THIS SNIPPET CAN BE COPIED

- Initialize MAGEMin with “ig” database
- Selects test 0, i.e., KLB-1 peridotite
- Set pressure and temperature
- Performs the calculation
- Deallocate memory

- “out” is a structure that stores all the information about the equilibrium. We will see how to create a vector of structure to stores multiple equilibrium point information

MAGEMin_C, first example

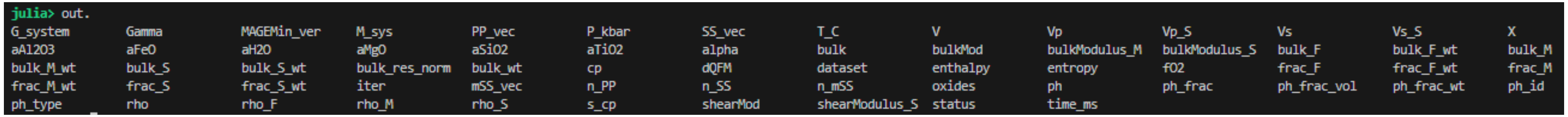
- Executing the previous set of commands gives a summary of the computed equilibrium



```
julia> out         =   single_point_minimization(P,T, data);
 Status             :            0
 Mass residual      : +2.44463e-14
 Rank               :            0
 Point              :            1
 Temperature        :   +800.00000       [C]
 Pressure           :     +8.00000       [kbar]

 SOL = [G: -797.749] (106 iterations, 87.90 ms)
 GAM = [-979.481429,-1774.103896,-795.262450,-673.746205,-375.079335,-917.592609,-829.966523,-1023.697396,-256.999589,-1308.291347]

 Phase :      spn      opx      cpx      ol
 Mode  :  0.02799  0.24227  0.14166  0.58807
```

Note that "GAM" stands for gamma and the array is the chemical potential of the oxides of the system

- To access the information store in "out" within the terminal simply type "out." then hit the "tab" key



```
julia> out.
G_system      Gamma         MAGEMin_ver   M_sys         PP_vec        P_kbar        SS_vec        T_C             V             Vp            Vp_S          Vs            Vs_S          X
aAl2O3        aFeO          aH2O          aMgO          aSiO2         aTiO2         alpha         bulk            bulkMod       bulkModulus_M bulkModulus_S bulk_F        bulk_F_wt     bulk_M
bulk_M_wt     bulk_S        bulk_S_wt     bulk_res_norm bulk_wt       cp            dQFM          dataset         enthalpy      entropy       fO2           frac_F        frac_F_wt     frac_M
frac_M_wt     frac_S        frac_S_wt     iter          mSS_vec       n_PP          n_SS          n_mSS           oxides        ph            ph_frac       ph_frac_vol   ph_frac_wt    ph_id
ph_type       rho           rho_F         rho_M         rho_S         s_cp          shearMod      shearModulus_S  status        time_ms
```

MAGEMin_C, output structure

- “out” structure is of type `MAGEMin_C.gmin_struct{Float64, Int64}`

- Some details about information stored in “out”

`out.dataset`	Dataset used for the calculation
`out.oxides`	List of oxides
`out.Gamma`	Chemical potential of oxides
`out.bulk`	Bulk rock composition in mol
`out.bulk_M`	Bulk composition of melt if any
`out.bulk_S`	Bulk composition of solid part if any
`out.ph`	Stable phases names
`out.ph_frac`	Stable phases fraction in mol
`out.frac_M`	Melt fraction in mol
`out.frac_S`	Solid fraction in mol
`…`	

`out.bulk_wt`	in wt
`out.bulk_M_wt`	in wt
`out.bulk_S_wt`	in wt
`out.ph_frac_wt`	in wt
`out.frac_M_wt`	in wt
`out.frac_S_wt`	in wt

MAGEMin_C, output structure

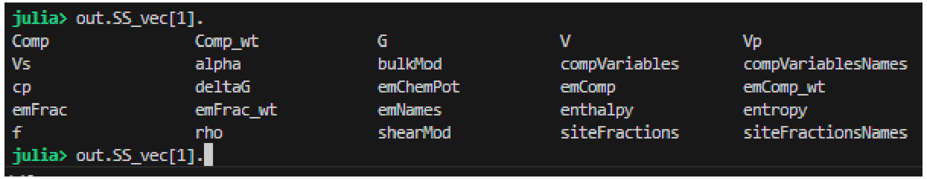
- Phases information can be access as:

`out.n_SS`	Number of stable solution phases
`out.n_PP`	Number of stable pure phases
`out.SS_vec`	Vector{MAGEMin_C.LibMAGEMin.SS_data}
`out.PP_Vec`	Vector{MAGEMin_C.LibMAGEMin.SS_data}

- You can access the details of solution phase 1 as:

```
out.SS_vec[1].
```

then hit the “tab” key



```
julia> out.SS_vec[1].
Comp             Comp_wt          G                V                Vp
Vs               alpha            bulkMod          compVariables    compVariablesNames
cp               deltaG           emChemPot        emComp           emComp_wt
emFrac           emFrac_wt        emNames          enthalpy         entropy
f                rho              shearMod         siteFractions    siteFractionsNames
julia> out.SS_vec[1].
```

- Then you can access any content of the solution model as

`out.SS_vec[1].Comp`	Composition of the solution in mol fraction
`out.SS_vec[1].emNames`	End-member names
`out.SS_vec[1].emFrac`	End-member fraction
…	

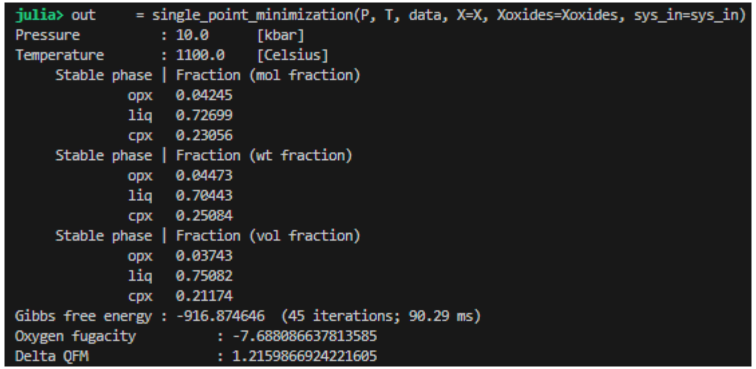
MAGEMin_C, custom bulk-rock composition

- Equilibrium calculation with custom bulk-rock composition is straigthforward

```julia
data     = Initialize_MAGEMin("ig", verbose=false);

P,T      = 10.0, 1100.0
Xoxides = ["SiO2"; "Al2O3"; "CaO"; "MgO"; "FeO"; "Fe2O3"; "K2O"; "Na2O"; "TiO2"; "Cr2O3"; "H2O"];
X        = [48.43; 15.19; 11.57; 10.13; 6.65; 1.64; 0.59; 1.87; 0.68; 0.0; 3.0];
sys_in   = "wt"
out      = single_point_minimization(P, T, data, X=X, Xoxides=Xoxides, sys_in=sys_in)
Finalize_MAGEMin(data)
```

THIS SNIPPET CAN BE COPIED

- Xoxides → string array of oxides names
- X → bulk-rock composition, provide either FeO,O or FeO,Fe2O3
- sys_in → system unit: mol or wt

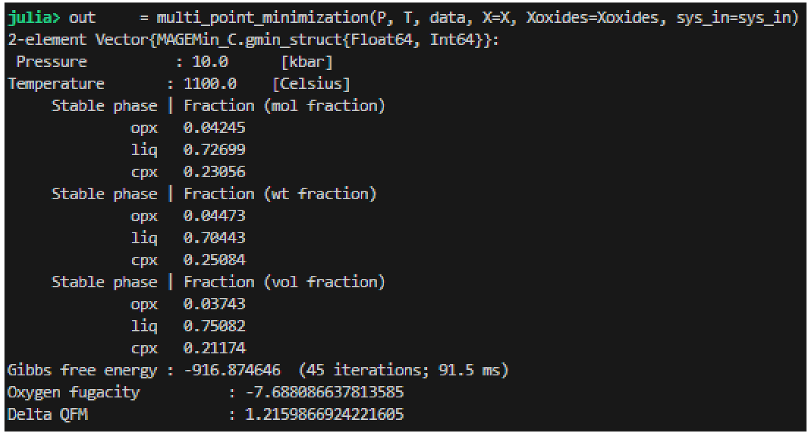


```
julia> out      = single_point_minimization(P, T, data, X=X, Xoxides=Xoxides, sys_in=sys_in)
Pressure          : 10.0      [kbar]
Temperature       : 1100.0    [Celsius]
     Stable phase | Fraction (mol fraction)
              opx   0.04245
              liq   0.72699
              cpx   0.23056
     Stable phase | Fraction (wt fraction)
              opx   0.04473
              liq   0.70443
              cpx   0.25084
     Stable phase | Fraction (vol fraction)
              opx   0.03743
              liq   0.75082
              cpx   0.21174
Gibbs free energy : -916.874646  (45 iterations; 90.29 ms)
Oxygen fugacity          : -7.688086637813585
Delta QFM                : 1.2159866924221605
```

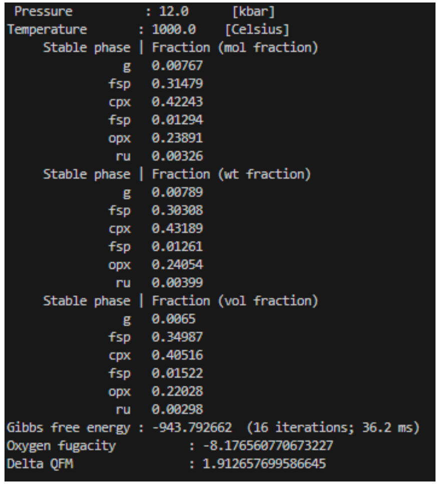
MAGEMin_C, parallel calculation

- When Julia is launched in parallel, one can perform equilibrium calculation of several points in parallel:

```julia
data    = Initialize_MAGEMin("ig", verbose=false);
P       = [10.0, 12.0]
T       = [1100.0, 1000.0]
Xoxides = ["SiO2"; "Al2O3"; "CaO"; "MgO"; "FeO"; "Fe2O3"; "K2O"; "Na2O"; "TiO2"; "Cr2O3"; "H2O"];
X1      = [48.43; 15.19; 11.57; 10.13; 6.65; 1.64; 0.59; 1.87; 0.68; 0.0; 3.0];
X2      = [49.43; 14.19; 11.57; 10.13; 6.65; 1.64; 0.59; 1.87; 0.68; 0.0; 0.0];
X       = [X1,X2]
sys_in  = "wt"
out     = multi_point_minimization(P, T, data, X=X, Xoxides=Xoxides, sys_in=sys_in)
Finalize_MAGEMin(data)
```



```
julia> out     = multi_point_minimization(P, T, data, X=X, Xoxides=Xoxides, sys_in=sys_in)
2-element Vector{MAGEMin_C.gmin_struct{Float64, Int64}}:
 Pressure          : 10.0      [kbar]
Temperature        : 1100.0    [Celsius]
     Stable phase | Fraction (mol fraction)
              opx   0.04245
              liq   0.72699
              cpx   0.23056
     Stable phase | Fraction (wt fraction)
              opx   0.04473
              liq   0.70443
              cpx   0.25084
     Stable phase | Fraction (vol fraction)
              opx   0.03743
              liq   0.75082
              cpx   0.21174
Gibbs free energy : -916.874646  (45 iterations; 91.5 ms)
Oxygen fugacity          : -7.688086637813585
Delta QFM                : 1.2159866924221605
```



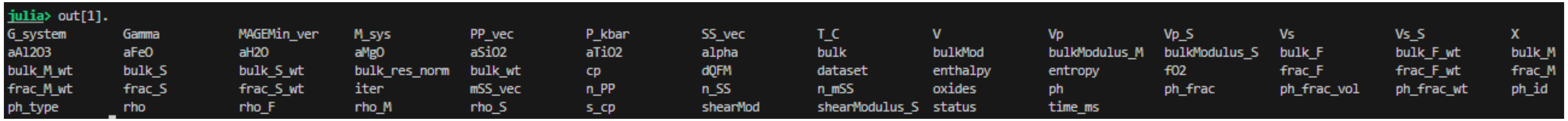
```
 Pressure          : 12.0      [kbar]
Temperature        : 1000.0    [Celsius]
     Stable phase | Fraction (mol fraction)
                g   0.00767
              fsp   0.31479
              cpx   0.42243
              fsp   0.01294
              opx   0.23891
               ru   0.00326
     Stable phase | Fraction (wt fraction)
                g   0.00789
              fsp   0.30308
              cpx   0.43189
              fsp   0.01261
              opx   0.24054
               ru   0.00399
     Stable phase | Fraction (vol fraction)
                g   0.0065
              fsp   0.34987
              cpx   0.40516
              fsp   0.01522
              opx   0.22028
               ru   0.00298
Gibbs free energy : -943.792662  (16 iterations; 36.2 ms)
Oxygen fugacity          : -8.176560770673227
Delta QFM                : 1.912657699586645
```

THIS SNIPPET CAN BE COPIED

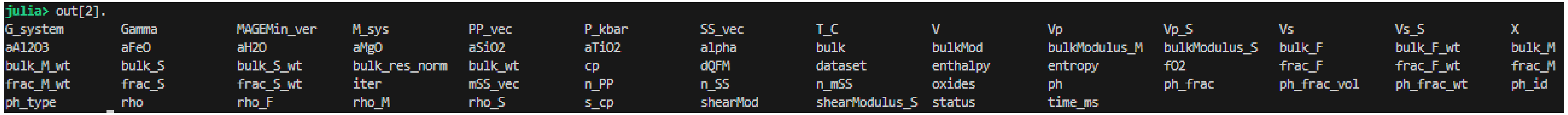
MAGEMin_C, parallel calculation: reading output

- For multi point minimization the outputs are store as `Vector{MAGEMin_C.gmin_struct{Float64, Int64}}`

- To access individual point information simply do



```
julia> out[1].
G_system    Gamma    MAGEMin_ver  M_sys          PP_vec   P_kbar  SS_vec    T_C              V          Vp              Vp_S            Vs           Vs_S         X
aAl2O3      aFeO     aH2O         aMgO           aSiO2    aTiO2   alpha     bulk             bulkMod    bulkModulus_M   bulkModulus_S   bulk_F       bulk_F_wt    bulk_M
bulk_M_wt   bulk_S   bulk_S_wt    bulk_res_norm  bulk_wt  cp      dQFM      dataset          enthalpy   entropy         fO2             frac_F       frac_F_wt    frac_M
frac_M_wt   frac_S   frac_S_wt    iter           mSS_vec  n_PP    n_SS      n_mSS            oxides     ph              ph_frac         ph_frac_vol  ph_frac_wt   ph_id
ph_type     rho      rho_F        rho_M          rho_S    s_cp    shearMod  shearModulus_S   status     time_ms
```

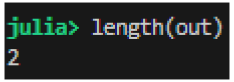


```
julia> out[2].
G_system    Gamma    MAGEMin_ver  M_sys          PP_vec   P_kbar  SS_vec    T_C              V          Vp              Vp_S            Vs           Vs_S         X
aAl2O3      aFeO     aH2O         aMgO           aSiO2    aTiO2   alpha     bulk             bulkMod    bulkModulus_M   bulkModulus_S   bulk_F       bulk_F_wt    bulk_M
bulk_M_wt   bulk_S   bulk_S_wt    bulk_res_norm  bulk_wt  cp      dQFM      dataset          enthalpy   entropy         fO2             frac_F       frac_F_wt    frac_M
frac_M_wt   frac_S   frac_S_wt    iter           mSS_vec  n_PP    n_SS      n_mSS            oxides     ph              ph_frac         ph_frac_vol  ph_frac_wt   ph_id
ph_type     rho      rho_F        rho_M          rho_S    s_cp    shearMod  shearModulus_S   status     time_ms
```

- The number of points can be retrieved using

```
length(out)
```

$\longrightarrow$



```
julia> length(out)
2
```

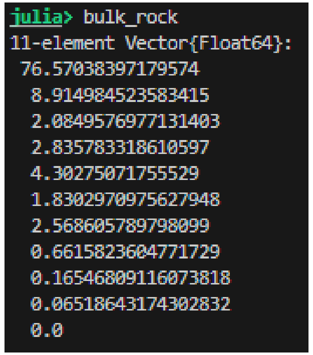
- Information of each minimization can otherwise be accessed as described in slides 7-8

MAGEMin_C, some useful commands

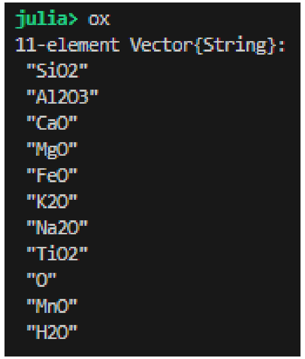
- Bulk-rock composition can be converted to MAGEMin system unit (mol) with

```
bulk_in_ox = ["SiO2"; "Al2O3"; "CaO"; "MgO"; "FeO"; "Fe2O3"; "K2O"; "Na2O"; "TiO2"; "MnO"; "H2O"];
bulk_in    = [69.64; 13.76; 1.77; 1.73; 4.32; 0.4; 2.61; 2.41; 0.80; 0.07; 0.0];
bulk_rock,ox  = convertBulk4MAGEMin(bulk_in,bulk_in_ox,"wt","mp");
```

Note that here, we convert for the “mp” (metapelite) database from wt to mol and transforming FeO,Fe2O3 to FeO,O



```
julia> bulk_rock
11-element Vector{Float64}:
 76.57038397179574
  8.914984523583415
  2.0849576977131403
  2.835783318610597
  4.30275071755529
  1.8302970975627948
  2.568605789798099
  0.6615823604771729
  0.16546809116073818
  0.06518643174302832
  0.0
```



```
julia> ox
11-element Vector{String}:
 "SiO2"
 "Al2O3"
 "CaO"
 "MgO"
 "FeO"
 "K2O"
 "Na2O"
 "TiO2"
 "O"
 "MnO"
 "H2O"
```

- Declaring an “array of output structure” to store results of several minimizations can be done with

```
n   = 10
out = Vector{MAGEMin_C.gmin_struct{Float64, Int64}}(undef, n)
```

Note that n can be any size

MAGEMin_C, fractional crystallization: definition

- In the following example we are going the fractionate a basaltic composition using the igneous database. We are also going to use “Plots” package to visualize the results
- First declare the database, initialize MAGEMin and provide bulk-rock composition, oxide list and system unit

```julia
using MAGEMin_C
using Plots

dtb     = "ig"
data    = Initialize_MAGEMin(dtb);

Xoxides = ["SiO2", "Al2O3", "CaO", "MgO", "FeO", "K2O", "Na2O", "TiO2", "O", "Cr2O3", "H2O"]
X       = [50.0, 8.7, 11.7, 12.14, 7.7, 0.2, 2.5, 1.0, 0.5, 0.01, 10.0]
sys_in  = "mol";
```

THIS SNIPPET CAN BE COPIED

Pressure →	`P       = 5.0`
Number of fractionation steps →	`n_steps = 64`
Starting temperature →	`Ts      = 1200.0`
Ending temperature →	`Te      = 600.0`
Define temperature range →	`T       = Array(range(Ts, stop=Te, length=n_steps))`
Declare output vector →	`out     = Vector{MAGEMin_C.gmin_struct{Float64, Int64}}(undef, n_steps)`

```julia
P       = 5.0
n_steps = 64
Ts      = 1200.0
Te      = 600.0
T       = Array(range(Ts, stop=Te, length=n_steps))
out     = Vector{MAGEMin_C.gmin_struct{Float64, Int64}}(undef, n_steps)
```

THIS SNIPPET CAN BE COPIED

MAGEMin_C, fractional crystallization: calculation

- In its simplest form, fractional crystallization uses the composition of the stable melt at a given temperature, as the starting bulk-rock composition at decreased temperature

```
for i in 1:n_steps
    out[i] = deepcopy(single_point_minimization(P, T[i], data, X=X, Xoxides=Xoxides, sys_in=sys_in))
    if "liq" in out[i].ph                                                          # If liquid phase is
         X = deepcopy(out[i].bulk_M)
    end
end
```

THIS SNIPPET CAN BE COPIED

- **In the above code:**
  1. We loop through all defined steps (64 as defined in previous slide)
  2. Perform the equilibrium calculation given the temperature array T[i] and the bulk composition X. Note that the result are stored in “out[i]”
  3. If “liq” is stable, we then update the bulk-rock composition X to be equal to the composition of the melt “out[i].bulk_M”
     otherwise, the bulk is unchanged

MAGEMin_C, fractional crystallization: data extraction

- Now that the calculation is performed, we need to be able to transform the saved data in a useful way to be plotted. The code snippet below, gives you some hints on how it can be done

```
frac_M      = [out[i].frac_M for i in 1:n_steps]
frac_M_tot  = accumulate(*, frac_M)

SiO2_id     = findfirst(out[1].oxides .== "SiO2
dry_id      = findall(out[1].oxides .!= "H2O")

SiO2_M_dry  = [ (out[i].bulk_M[SiO2_id] / sum(out[i].bulk_M[dry_id])*100.0) for i in 1:n_steps]

rho_M       = [ (out[i].rho_M) for i in 1:n_steps]
rho_M[rho_M .== 0.0] .= NaN;
```

THIS SNIPPET CAN BE COPIED

- **In the code above, we extract and clean several information**
  1. The fraction of stable melt in mol
  2. The accumulated stable melt fraction with respect to the starting mol quantity
  3. We retrieve the array index of SiO2 and the indices of all oxides excluding water, this for anhydrous normalization
  4. We compute the SiO2 content of the melt on an anhydrous basis
  5. We retrieve the melt density and set to NaN the values equal to 0.0 (when melt is not stable)

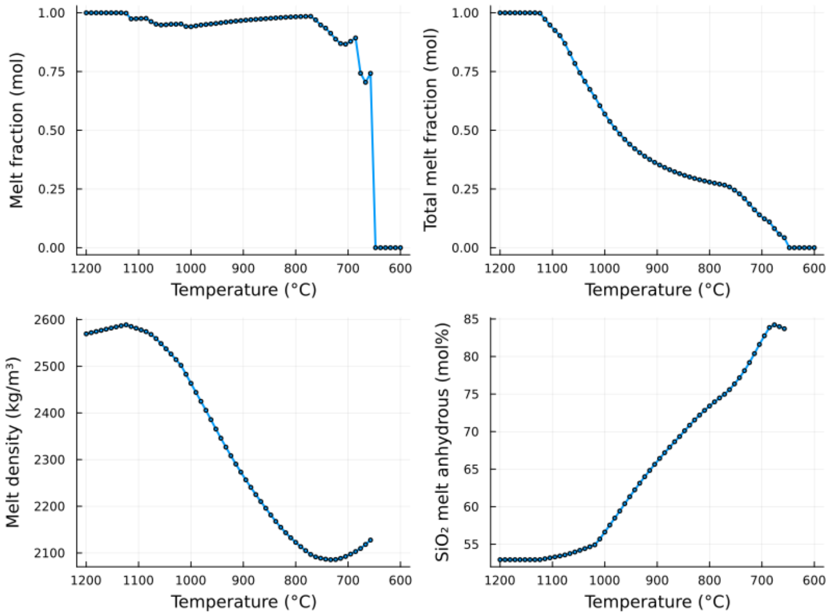
MAGEMin_C, fractional crystallization: visualization

- Now that some data have been converted, we can visualize it. Here is provided one example of visualization using Plots, you are of course welcome to use any other package

```
p1 = plot(T,frac_M, xflip=true, xlabel="Temperature (°C)", marker = :circle, markersize = 2, lw=2, ylabel="Melt fraction (mol)", legend=false)
p2 = plot(T,frac_M_tot, xflip=true, xlabel="Temperature (°C)", marker = :circle, markersize = 2, lw=2, ylabel="Total melt fraction (mol)", legend=false)
p3 = plot(T,rho_M, xflip=true, xlabel="Temperature (°C)", marker = :circle, markersize = 2, lw=2, ylabel="Melt density (kg/m³)", legend=false)
p4 = plot(T,SiO2_M_dry, xflip=true, xlabel="Temperature (°C)", marker = :circle, markersize = 2, lw=2, ylabel="SiO₂ melt anhydrous (mol%)", legend=false)

fig = plot(p1, p2, p3, p4, layout=(2, 2), size=(800, 600))
savefig(fig,"frac_crystallization.png")
```

THIS SNIPPET CAN BE COPIED

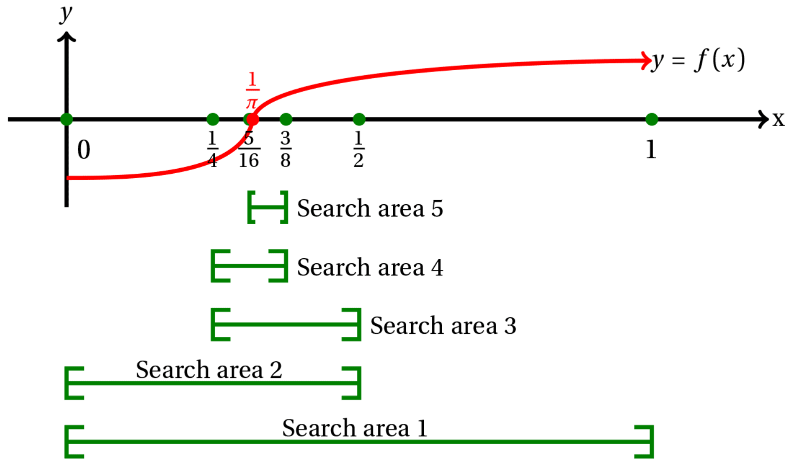
- Which yields



MAGEMin_C, advanced example: isentropic path

- Isentropic paths (Adiabatic decompression) can be generated directly in MAGEMinApp, but here we are going to see how this can be achieved using MAGEMin_C
- The objective is to provide a starting PT condition, compute the entropy of the system, then gradually decrease pressure while keeping entropy constant and computing the corresponding temperature

- Because the temperature at decreased pressure that keeps entropy constant is unknown, we need a strategy to constrain it. Here we are going to use the bisection method.
- The idea is to reformulate the problem as a root finding problem and use the bisection method as described below:



- f(x) = Sref – Sx
  where x is temperature
  Sref is entropy of reference
  Sx is the entropy at tentative temperature x

MAGEMin_C, isentropic path: definition

ProgressMeter to display a progress bar

Igneous database
Initialize MAGEMin while deactivating verbose
Select predefined KLB-1 test
Load predefined test

Mantle potential temperature
Adiabatic gradient
Starting depth
Mantle density

Starting temperature
Starting pressure
Ending pressure
Number of steps
Maximum number of bisection iterations
Tolerance in K

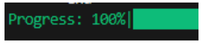
Pressure range for the isentropic path
Output structure Vector
Temporary output structure
Compute reference point at Ts and Ps
Reference entropy

```julia
using MAGEMin_C
using Plots
using ProgressMeter

dtb         = "ig"
data        = Initialize_MAGEMin(dtb,verbose=-1)
test        = 0          # KLB-1
data        = use_predefined_bulk_rock(data, test)

MPT         = 1350.0
adiabat     = 0.55
Depth       = 100.0
rho_Mantle  = 3300.0

Ts          = MPT + adiabat*Depth
Ps          = Depth*1e3*9.81*rho_Mantle/1e5/1e3
Pe          = 0.001
n_steps     = 32
n_max       = 32
tolerance   = 0.1

P           = Array(range(Ps, stop=Pe, length=n_steps))
out         = Vector{MAGEMin_C.gmin_struct{Float64, Int64}}(undef, n_steps)
out_tmp     = MAGEMin_C.gmin_struct{Float64, Int64};
out[1]      = deepcopy( single_point_minimization(Ps,Ts, data));
Sref        = out[1].entropy
```

THIS SNIPPET CAN BE COPIED

MAGEMin_C, isentropic path: calculation

- This code snippet uses the bisection method to find the temperature fitting the entropy of reference within given tolerance

- The command “@showprogress” place before the pressure loop allow to display the progress bar in the Julia terminal 

```
Progress: 100%|
```

- **a** and **b** define the starting temperature bounds
  n is the number of bisection
  c is the bisected temperature
  result is the misfit (f(x) = Sref – Sx)
  sign_c is the sign of the misfit a T = **c**

- If the bounds distance is < tolerance, then solution is accepted
  else, the bounds are updated

```julia
@showprogress for j = 2:n_steps
        a            = out[j-1].T_C - 50.0
        b            = out[j-1].T_C
        n            = 1
        conv         = 0
        n            = 0
        sign_a       = -1

        while n < n_max && conv == 0
            c        = (a+b)/2.0
            out_tmp  = deepcopy( single_point_minimization(P[j],c, data));
            result   = out_tmp.entropy - Sref
            sign_c   = sign(result)

            if abs(b-a) < tolerance
                conv = 1
            else
                if  sign_c == sign_a
                    a = c
                    sign_a = sign_c
                else
                    b = c
                end
            end
            n += 1
        end
        out[j] = deepcopy(out_tmp)
end

Finalize_MAGEMin(data)
```

THIS SNIPPET CAN BE COPIED

MAGEMin_C, isentropic path: data extraction

- Similarly to the factional crystallization example, data output need to be reformatted.

```
S            = [out[i].entropy for i in 1:n_steps]
frac_M       = [out[i].frac_M for i in 1:n_steps]
frac_M[frac_M .== 0.0] .= NaN
T            = [out[i].T_C for i in 1:n_steps]
SiO2_id      = findfirst(out[1].oxides .== "SiO2")
dry_id       = findall(out[1].oxides .!= "H2O")
SiO2_M_dry   = [ (out[i].bulk_M[SiO2_id] / sum(out[i].bulk_M[dry_id])*100.0) for i in 1:n_steps];
rho_M        = [ (out[i].rho_M) for i in 1:n_steps];
rho_M[rho_M .== 0.0] .= NaN;
```

THIS SNIPPET CAN BE COPIED

- **In the code above, we extract and clean several information**
  1. The entropy of all points for checking
  2. The fraction of stable melt in mol and we set to NaN the values equal to 0.0 (when melt is not stable)
  3. We retrieve the array index of SiO2 and the indices of all oxides excluding water, this for anhydrous normalization
  4. We compute the SiO2 content of the melt on an anhydrous basis
  5. We retrieve the melt density and set to NaN the values equal to 0.0 (when melt is not stable)

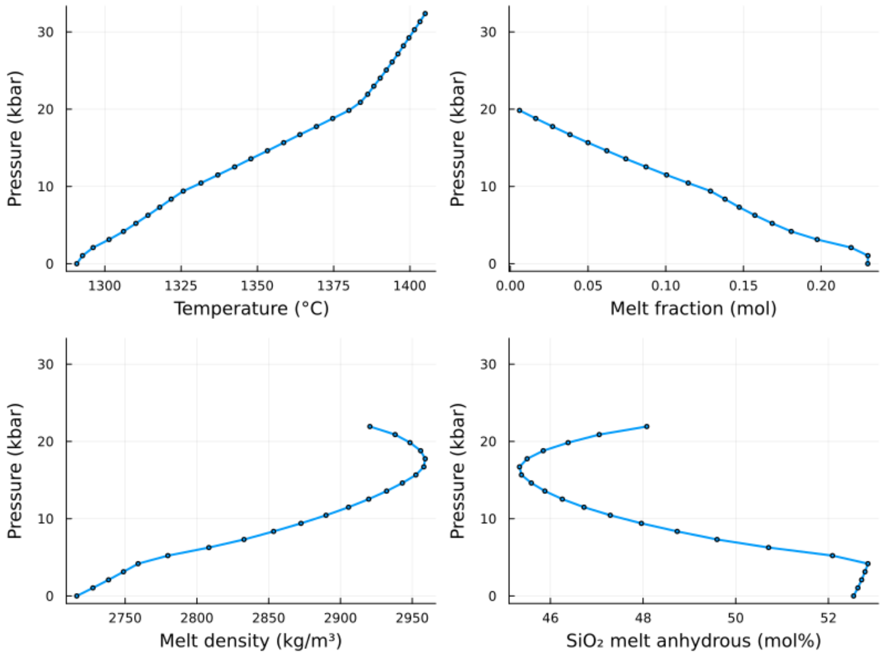
MAGEMin_C, isentropic path: visualization

- Visualization is achieved in a similar way as for the fractional crystallization example

```
p1          = plot(T,P, xlabel="Temperature (°C)", marker = :circle, markersize = 2, lw=2, ylabel="Pressure (kbar)", legend=false)
p2          = plot(frac_M,P, xlabel="Melt fraction (mol)", marker = :circle, markersize = 2, lw=2, ylabel="Pressure (kbar)", legend=false)
p3          = plot(rho_M,P, xlabel="Melt density (kg/m³)", marker = :circle, markersize = 2, lw=2, ylabel="Pressure (kbar)",         legend=false)
p4          = plot(SiO2_M_dry,P, xlabel="SiO₂ melt anhydrous (mol%)", marker = :circle, markersize = 2, lw=2, ylabel="Pressure (kbar)",  legend=false)

fig = plot(p1, p2, p3, p4, layout=(2, 2), size=(800, 600))
savefig(fig,"isentropic_path.png")
```

THIS SNIPPET CAN BE COPIED

- Which yields


Pressure (kbar)
Temperature (°C)
Melt fraction (mol)
Melt density (kg/m³)
SiO₂ melt anhydrous (mol%)