```
In[•]:= ClearAll["Global`*"]
```

```
In[•]:= (* Set the notebook directory to the local directory *)
SetDirectory[NotebookDirectory[]];
(* Check that this is the directory *)
NotebookDirectory[]
```

Overview of the program

```
In[•]:= (*




This notebook computes the "quasinormal modes" associated with perturbations of
  holographic superconductors. This simple example illustrates a few numerical
  techniques including pseudospectral methods to solve differential equations
  as well as finding eigenvalues. This is a version of the numerics in https://
 arxiv.org/pdf/2212.10410 which rescales the U(1) gauge field A_M→
  A_M/Q and the complex scalar field ψ→ψ_M/Q with Q→∞. In this limit,
the spacetime is fixed and only the matter fields have non-
 trivial behavior. This is equivalent to suppressing energy and
  momentum fluctuations in the superconductor and only allowing
  charge fluctuations. The hydrodynamic modes ω(k) can be matched
  exactly including both real and imaginary parts in the limit k<<1,
however for the sake of readability, I will only match the velocity of
 the second sound mode: lim_{k→0}ω(k). To obtain the imaginary components,
one needs the low frequency response functions and then uses Kubo formulae--
 the numerical techniques for these are very similar to what is shown here,
so I have omitted this step. If you would like to see how that works,
let me know and I will append it to the program.




*)

(*




                There are five main parts to the program which can each be run
                 independently or the whole notebook can be run from the top.

                  1.) In the first part we derive the equations
```

of motion describing holographic superconductors in equilibrium and then subject to a linearized perturbation.

2.) In the second part we solve the equations of motion for the background solution. The asymptotic behavior of these solutions defines sources and expectation values for the electric current < J^{μ}> and the order parameter <$\psi^{*}\psi$> . This is a boundary value problem for a coupled set of ordinary differential equations.

3.) In the third part we solve the linearized equations of subject to the boundary condition that the sources vanish. Since response functions are formally given by $G^{R}_{OO}(\omega,k) = \frac{\delta\langle O\rangle}{\delta s_0}$ where s_0 is the source for the operators <O>, solutions to these equations give the poles of the response functions. They exist at limited values of $\omega(k)$ which define the dispersion relations of hydrodynamic fluctuations in a superconductor. Numerically, this means that we must find the $\omega(k)$, i.e. we solve an eigenvalue problem. The matrices are dense so there is no way to make diagonalization faster and Mathematica' s built in eigensolver is sufficiently efficient. However, in step 4, we illustrate a way to isolate an individual eigenvalue and increase precision and accuracy.

4.) We show how to efficiently increase accuracy for a single eigenvalue.

5.) Finally, we compare to the predictions of the hydrodynamic theory and show that the results agree.

*)

In[•]:=
```
(* More details on part 2.

  To solve the ODEs, we will use a version of Newton's method. The
 essential idea is that we have an equation we want to solve,
which we may write E_j[S^i] = 0 for some configuration of the fields S^i,
but we are at a different value X^i(t=0). As with Newton's method of root finding,
from step t to step t+1,
we choose the next X^i(t+1) such that E_j'[X^i(t)](X^i(t+1)-X^i(t)) =
 -E_j[X^i(t)]. The fields X^i are a vector so this
  can be solved with built in linear algebra tools.

   Mathematica is competitive with other
  programming languages for linear algebra, see https://
 reference.wolfram.com/language/tutorial/LinearAlgebraInMathematicaOverview.html.

     At each step, then, we solve M^ij(t)δX^j(t)=-E^i(t) with M^ij(t) =
  δE^i/δX^j(t) until we reach a point where δX^i is below some threshold.
*)
```

In[∘]:= (* More details on part 3.

The linearized equations of motion have the form $\sum_n \omega^n C^i{}_j(n)\,\delta X^j = 0$, where $C^i{}_j$ are differential operators (that include the boundary conditions) acting on the linearized fields δX^j and ω is the frequency of the plane wave perturbation. A simple example which has only one power of ω can be written,

$$C^i{}_j(0)\,\delta X^j = -\omega C^i{}_j(1)\,\delta X^j,$$

which has the form of a generalized eigenvalue problem with matrices $C^i{}_j(0)$ and $-C^i{}_j(1)$. Generalized eigenvalue equations can be solved efficiently with Mathematica. When there are more powers of ω, the equations still have the form of a generalized eigenvalue problem, but in terms of a modified eigenvector. For instance, if the largest power of ω is 2, then,

$$\begin{pmatrix} C^i{}_j(1) & C^i{}_j(0) \\ -I & 0 \end{pmatrix}\begin{pmatrix} \omega\delta X \\ \delta X \end{pmatrix} = -\omega \begin{pmatrix} C^i{}_j(2) & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} \omega\delta X \\ \delta X \end{pmatrix},$$

returns the same linearized equation but increases the size of the matrices by a power of 4. If higher powers of ω appeared we would continue this process,

$$\begin{pmatrix} C^i{}_j(\text{nmax}-1) & C^i{}_j(\text{nmax}-2) & \ldots & C^i{}_j(0) \\ -I & 0 & \ldots & 0 \\ - & -I & \ldots & 0 \\ 0 & 0 & \ldots & 0 \end{pmatrix}\begin{pmatrix} \omega^{n-1}\delta X \\ \omega^{n-2}\delta X \\ \ldots \\ \delta X \end{pmatrix} = -\omega \begin{pmatrix} C^i{}_j(\text{nmax}) & 0 & \ldots & 0 \\ 0 & I & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & I \end{pmatrix}\begin{pmatrix} \omega^{n-1}\delta X \\ \omega^{n-2}\delta X \\ \ldots \\ \delta X \end{pmatrix}.$$

*)

In[•]:=
```
(* More details on part 4.

 In part 4, we increase the grid size to get better accuracy. However,
this drastically increases the size of the matrices we would need to
 diagonalize. In general, we care about only the lowest lying eigenvalues,
so diagonalizing the full matrix is overkill. The most efficient route is to
 promote ω to a function of z, ω→ω[z], with an equation of motion ω'[z] = 0,
and then use Newton-Raphson. When we do this,
it is important to fix a normalization for the eigenvector,
though the choice is arbitrary. We will choose
 that the linearized fields sum to one on the horizon.

*)
```

In[•]:=
```
(* Other notes: There are ways to optimize these types of programs via compiled
   functions which can include arbitrary precision computations. However,
these tend to slightly increase overhead for the problem presented
 here so instead I just construct the numerical versions of the
 differential equations using replacement rules which resulted
 in a more efficient and readable code for this problem. *)
```

1.) Deriving the equations of motion

2.) Background field configurations via Newton-Raphson

3.) Quasinormal modes or hydrodynamic spectrum

In[•]:=
```
(* Make sure variables don't have any values stored in memory*)
```

In[•]:=
```
Clear[kk, zmin, zmax]
```

Numerical Parameters

In[•]:=
```
(* If we start here,
useful to redefine parameters for the numerics. Start with precision *)
```

In[•]:=
```
{zmin, zmax} = {0, 1};
```

In[•]:=
```
mp = 60;
$MinPrecision = mp;
```

In[•]:=
```
(* Set threshold for "0" < 10^-mp *)
```

In[•]:=
```
chopmin = 10^-100;
```

```
In[•]:= (* Define number of grid points and min and max values. Since we
         will need a background solution, here we import the solution. *)

In[•]:= {Q, λdata, D0λ, D1λ, D2λ, μtab, μstab, ϕvectab, Axvectab, ηvectab} =
          Import[NotebookDirectory[] <> "Data/probe_background_4d_sols.mx"];

In[•]:= NumGridPoints = Length[λdata] - 1;

In[•]:= Nμ = Length[μtab] - 1;
        Nμs = Length[μstab] - 1;
        {μmin, μmax} = {μtab[[1]], μtab[[Nμ + 1]]};
        {μsmin, μsmax} = {μstab[[1]], μstab[[Nμs + 1]]};

In[•]:= (* We need to discretize the wavevectors. Can do a Chebyshev grid again. It is
          good to solve for at least 2 different k values so that you can check ω ~ vk +
         O(k^2) for the second sound mode. More k vectors is even better,
        but time consuming. Below, we will do this using a faster method. *)
        Numk = 1;
        {kmin, kmax} = {1/100, 1/10};
        ktab = N[Table[(kmax + kmin)/2 - (kmax - kmin)/2 * Cos[π * j / Numk], {j, 0, Numk}], mp];
        eigs = Table[0, {i, 1, Numk + 1}];
```

Initialize EQs

Linearize EQs

Find eigenvalues

```
In[•]:= (* Can choose a particular solution, or could loop over solutions,
        though would need to change the structure of the eigs array. *)
        {μStart, μFinish} = {8, 8};
        {μsStart, μsFinish} = {7, 7};
        μscount = μsStart;
        μcount = μStart;
        kcount = 1;

In[•]:= nb = CreateDocument["", WindowSize → {Scaled[1 / 5], Scaled[1 / 2]}];
        While[μscount ≤ μsFinish,

          μcount = μStart;

          While[μcount ≤ μFinish,

           kcount = 1;
```

```
While[kcount ≤ Numk + 1,

 NotebookWrite[nb,
  Cell[BoxData@RowBox[{"kcount =", kcount, "timing = ", Timing[


        μ = μtab⟦μcount⟧;
        μs = μstab⟦μscount⟧;
        ηvec = ηvectab⟦μscount, μcount⟧;
        ϕvec = ϕvectab⟦μscount, μcount⟧;
        Axvec = Axvectab⟦μscount, μcount⟧;
        kk = ktab⟦kcount⟧;


        Do[coeffargs[i] = {z → λdata⟦i⟧,
           ϕ''[z] → D2λ⟦i⟧.ϕvec, ϕ'[z] → D1λ⟦i⟧.ϕvec, ϕ[z] → ϕvec⟦i⟧,
           Ax''[z] → D2λ⟦i⟧.Axvec, Ax'[z] → D1λ⟦i⟧.Axvec, Ax[z] → Axvec⟦i⟧,
           η''[z] → D2λ⟦i⟧.ηvec, η'[z] → D1λ⟦i⟧.ηvec, η[z] → ηvec⟦i⟧
          }, {i, 1, NumGridPoints + 1}];


        M = ArrayFlatten[Table[Sum[Flatten[Table[Piecewise[{{linBCcoeffs[m,
                    1, n, dd] /. (coeffargs[1] /. {z → zmin}), i == 1},
                 {linBCcoeffs[m, 2, n, dd] /. (coeffargs[NumGridPoints + 1] /.
                     {z → zmax}), i == NumGridPoints + 1}, {linEQcoeffs[m, n,
                    dd] /. coeffargs[i], i ≠ 1 && i ≠ NumGridPoints + 1}}],
               {i, 1, NumGridPoints + 1}]] × Drmatrices⟦dd⟧, {dd, 1,
             Length[Drmatrices]}], {m, 1, Length[EQs]}, {n, 1, Length[EQs]}]];

        MQNM0 = M /. {ω → 0};
        Mdim = Length[MQNM0⟦1⟧];
        MQNM1 = D[M, ω] /. {ω → 0};
        MQNM2 = (1 / 2) * D[M, {ω, 2}] /. {ω → 0};
        Amat = ArrayFlatten[{{MQNM2, 0 * IdentityMatrix[Mdim]},
           {0 * IdentityMatrix[Mdim], IdentityMatrix[Mdim]}}];
        Bmat = ArrayFlatten[
          {{MQNM1, MQNM0}, {-IdentityMatrix[Mdim], 0 * IdentityMatrix[Mdim]}}];
        Clear[M, MQNM0, MQNM1, MQNM2, Mdim];

        eigs⟦kcount⟧ = Chop[Eigensystem[{Bmat, -Amat}], chopmin];


       ]⟦1⟧}], "Output"]];
```

```
      kcount++;

     ];

     μcount++;

    ];

    μscount++;

   ];
  NotebookClose[nb];
```

```
In[∘]:= Export[NotebookDirectory[] <> "Data/probe_eigenvalues_and_eigenvectors_4d.mx",
          {μsStart, μsFinish, μStart, μFinish, ktab, eigs}];
```

4.) Optional: Increasing accuracy

```
In[∘]:= (* If we like, we can increase the accuracy of a qnm solution. The simplest way
          to do this is to promote ω to a function of z and use Newton-Raphson,
        since this is much more efficient than diagonalizing a matrix. We
         will increase the size of the grid using our previous
         solutions as seeds. First we will do the background*)
```

Background field configurations via Newton-Raphson

Choose an index for μ and μs

```
In[∘]:= (* Since we only looked at one choice of μ and μs for ω above,
        we will just use those*)
```

```
In[∘]:= {μsind, μind} = {μsStart, μStart};
```

Define Chebyshev grid and derivative matrices on the larger grid

```
In[∘]:= (* Increase grid size. It is good to do this little
         by little since the eigenfunctions are highly oscillatory,
        interpolation will introduce substantial error *)
```

```
In[∘]:= NumGridPoints = 60;
        {zmin, zmax} = {0, 1};
```

```
In[]:= (* grid points λ *)
λdata = N[Table[(zmax + zmin)/2 - (zmax - zmin)/2 * Cos[π * j / NumGridPoints],
    {j, 0, NumGridPoints}], mp];(*Chebyshev Table*)
aλ = N[Table[Product[If[j == k, 1, (λdata[[j]] - λdata[[k]])], {k, 1, NumGridPoints + 1}],
    {j, NumGridPoints + 1}], mp];
D1λ = Table[If[i == j, Sum[If[k == j, 0, 1/(λdata[[j]] - λdata[[k]])], {k, 1, NumGridPoints + 1}],
    aλ[[i]]/(aλ[[j]] (λdata[[i]] - λdata[[j]]))],
   {i, 1, NumGridPoints + 1}, {j, 1, NumGridPoints + 1}];
Clear[aλ];
D2λ = D1λ.D1λ;
D0λ = IdentityMatrix[NumGridPoints + 1];
Drmatrices = {D2λ, D1λ, D0λ};
(* test to check derivatives *)
Max[Table[3 λdata[[i]]^2, {i, 1, NumGridPoints + 1}] -
   D1λ.Table[λdata[[i]]^3, {i, 1, NumGridPoints + 1}] // Chop]
Max[Table[6 λdata[[i]], {i, 1, NumGridPoints + 1}] -
   D2λ.Table[λdata[[i]]^3, {i, 1, NumGridPoints + 1}] // Chop]

Out[]= 0

Out[]= 0
```

Numerical Parameters

```
In[]:= mp = 60;
$MinPrecision = mp;

In[]:= chopmin = 10^-100;

In[]:= (* Import the lower accuracy backgrounds*)

In[]:= {Q, λdataLowPrec, D0λLowPrec, D1λLowPrec, D2λLowPrec,
    μtab, μstab, ϕvectabLowPrec, AxvectabLowPrec, ηvectabLowPrec} =
   Import[NotebookDirectory[] <> "Data/probe_background_4d_sols.mx"];

In[]:= (* Import the lower accuracy eigenvectors and eigenvalues*)

In[]:= {μsStart, μsFinish, μStart, μFinish, ktabLowPrec, eigsLowPrec} =
   Import[NotebookDirectory[] <> "Data/probe_eigenvalues_and_eigenvectors_4d.mx"];

In[]:= (* Make sure there is an eigenvector and eigenvalue at the desired index,
else choose the first value for which there is one *)
```

```
In[∘]:= μsind = If[μsStart ≤ μsind ≤ μsFinish, μsind, μsStart];
        μind = If[μStart ≤ μind ≤ μFinish, μind, μStart];
```

Initialize EQs

```
In[∘]:= fieldredefa = {η[z] → z^2 * η[z], ϕ[z] → ϕ[z] (1 - z), f[z] → 1 - z^3};
        fieldredef = Flatten[{fieldredefa, D[fieldredefa, z], D[fieldredefa, {z, 2}]}];
        Clear[fieldredefa]
```

```
In[∘]:= backgroundEQs =
          Import[NotebookDirectory[] <> "EQs/probe_4d_backgroundEQs.mx"] /. fieldredef //
            Expand // Simplify;
        Clear[fieldredef]
```

```
In[∘]:= Series[backgroundEQs, {z, 0, -1}] // Simplify
        Series[backgroundEQs /. {z → 1 - z}, {z, 0, -1}] // Simplify
```

```
Out[∘]= {O[z]^3, O[z]^4, O[z]^4}
```

```
Out[∘]= {O[z]^1, O[z]^1, O[z]^0}
```

```
In[∘]:= EQs =
          ({backgroundEQs[[1]]/(z^3 (1 - z)), backgroundEQs[[2]]/(z^4 (1 - z)), backgroundEQs[[3]]/z^4}) // Expand // Simplify //
            PowerExpand // Simplify;
```

```
In[∘]:= {bc1, bc2, bc3} = EQs /. {z → 1} // Simplify;
```

```
In[∘]:= BCs = {{η'[zmin], bc1}, {ϕ[zmin] - μ, bc2}, {Ax[zmin] - μs, bc3}} // Simplify;
```

```
In[∘]:= Clear[bc1, bc2, bc3, backgroundEQs]
```

Linearize EQs

```
In[∘]:= (* Setting up the Newton-Raphson equations*)
```

```
In[∘]:= fieldRep = {η → η + ϵs * δη, ϕ → ϕ + ϵs * δϕ, Ax → Ax + ϵs * δAx};
        linBackEQ = ∂ϵs (EQs /. fieldRep) /. ϵs → 0;
        linBackBC = ∂ϵs (BCs /. fieldRep) /. ϵs → 0;
        Clear[fieldRep];

        linFieldList = {δη[z], δϕ[z], δAx[z]};
        (*dlinFieldList⟦choose a function⟧⟦choose a derivative⟧*)
        dlinFieldList =
          Table[{∂z,z linFieldList⟦fun⟧, ∂z linFieldList⟦fun⟧, linFieldList⟦fun⟧},
           {fun, 1, Length[linFieldList]}];
        Clear[linFieldList]
        (* Note: must change argument for boundary conditions from z to zmin or zmax *)
        boundrules = {z → zmin, z → zmax};
        Do[linEQcoeffs[eq, fun, der] = D[linBackEQ⟦eq⟧, dlinFieldList⟦fun⟧⟦der⟧],
          {eq, 1, Length[EQs]}, {fun, 1, Length[EQs]}, {der, 1, Length[dlinFieldList⟦eq⟧]}];
        (*BCc⟦BC_i⟧⟦min or max⟧⟦choose a function⟧⟦derivative⟧*)
        Do[linBCcoeffs[eq, bound, fun, der] =
           ∂(dlinFieldList⟦fun⟧⟦der⟧/.boundrules⟦bound⟧) linBackBC⟦eq⟧⟦bound⟧, {eq, 1, Length[EQs]},
          {bound, 1, 2}, {fun, 1, Length[EQs]}, {der, 1, Length[dlinFieldList⟦eq⟧]}];

In[∘]:= Clear[linBackBC, linBackEQ, dlinFieldList, boundrules]
```

Seeds

Newton-Raphson loop

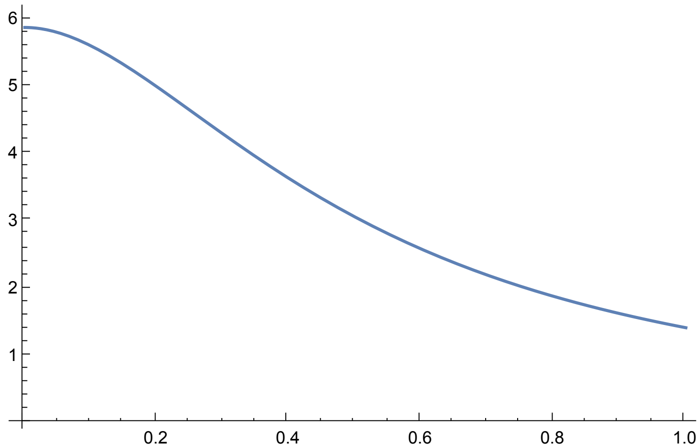
```
In[∘]:= (* Define thresholds which will halt the evaluation if exceeded*)

In[∘]:= Error = 10^-(mp/2);
        ErrorMax = 1000000;

In[∘]:= nb = CreateDocument["", WindowSize → {Scaled[1/5], Scaled[1/2]}];


        μ = μtab⟦μind⟧;
        μs = μstab⟦μsind⟧;

        ϵp = (ErrorMax - Error) / 2;
        count = 0;

        NotebookWrite[nb,
          Cell[BoxData@RowBox[{"μs # ", μscount, ".) μ # ", μcount, ".) ", Timing[

                While[ϵp > Error && ϵp < ErrorMax,
```

```
(* rules for replacing functions in the equations *)
Do[coeffargs[i] = {z → λdata⟦i⟧,
   ϕ''[z] → D2λ⟦i⟧.ϕvec, ϕ'[z] → D1λ⟦i⟧.ϕvec, ϕ[z] → ϕvec⟦i⟧,
   Ax''[z] → D2λ⟦i⟧.Axvec, Ax'[z] → D1λ⟦i⟧.Axvec, Ax[z] → Axvec⟦i⟧,
   η''[z] → D2λ⟦i⟧.ηvec, η'[z] → D1λ⟦i⟧.ηvec, η[z] → ηvec⟦i⟧
  }, {i, 1, NumGridPoints + 1}];

(* Construct the derivative matrix *)
M = ArrayFlatten[Table[Sum[Flatten[Table[Piecewise[{{linBCcoeffs[m,
          1, n, dd] /. (coeffargs[1] /. {z → zmin}), i == 1},
        {linBCcoeffs[m, 2, n, dd] /. (coeffargs[NumGridPoints + 1] /.
            {z → zmax}), i == NumGridPoints + 1}, {linEQcoeffs[m, n, dd] /.
           coeffargs[i], i ≠ 1 && i ≠ NumGridPoints + 1}}],
      {i, 1, NumGridPoints + 1}]] × Drmatrices⟦dd⟧, {dd, 1, Length[
     Drmatrices]}], {m, 1, Length[EQs]}, {n, 1, Length[EQs]}]];

(* Construct the value of the
 eoms given the current configuration of the fields. *)
Etot = Chop[Flatten[Parallelize[Table[Piecewise[
     {{BCs⟦m, 1⟧ /. (coeffargs[1] /. {z → zmin}), i == 1}, {BCs⟦m, 2⟧ /.
        (coeffargs[NumGridPoints + 1] /. {z → zmax}), i == NumGridPoints +
         1}, {EQs⟦m⟧ /. coeffargs[i], i ≠ 1 && i ≠ NumGridPoints + 1}}],
    {m, 1, Length[EQs]}, {i, 1, NumGridPoints + 1}]]], chopmin];

(* Solve for the change in the fields *)
deltaFields = Chop[LinearSolve[M, -Etot], chopmin];

(* Compute the norm of deltaFields to keep track of convergence *)
ϵp = Norm[deltaFields, ∞];

If[NumberQ[ϵp], NotebookWrite[nb,
  Cell[BoxData@RowBox[{"ϵp= ", ϵp}], "Output"]], Break[]];

(* If the norm is too large, introduce friction. *)
friction = If[ϵp < 1, 1, 1 / 10];

combinedFields = combinedFields + friction * deltaFields;
(* partition the solution
 for the combined fields into individual fields *)
{ηvec, ϕvec, Axvec} = Partition[combinedFields, NumGridPoints + 1];

Clear[friction, deltaFields];

count++;
```

```
            ];

        ]⟦1⟧, ", # Iteration = ", count}], "Output"]];

NotebookClose[nb];

(* Example plot of a background field *)
```

In[∘]:=
```
ListLinePlot[Table[{λdata⟦i⟧, ηvec⟦i⟧}, {i, 1, NumGridPoints + 1}]]
```

Out[∘]=



In[∘]:=
```
(* Export solutions,
include Chebyshev grid parameters so we don't duplicate efforts *)
```

In[∘]:=
```
Export[NotebookDirectory[] <> "Data/probe_background_4d_sols_increasedAccuracy.mx",
   {μsind, μind, Q, λdata, D0λ, D1λ, D2λ, ϕvec, Axvec, ηvec}];
```

Quasinormal mode via Newton-Raphson. Want to track superfluid second sound.

In[∘]:=
```
(* We will now scan a larger set of wavevectors zooming in on a single mode. *)
```

Numerical Parameters

In[∘]:=
```
(* If we start here,
useful to redefine parameters for the numerics. Start with precision *)
```

In[∘]:=
```
mp = 60;
$MinPrecision = mp;
```

In[∘]:=
```
(* Set threshold for "0" < 10^-mp *)
```

In[∘]:=
```
chopmin = 10^-100;
```

In[∘]:=
```
(* Import higher accuracy background *)
```

In[∘]:=
```
{μsind, μind, Q, λdata, D0λ, D1λ, D2λ, ϕvec, Axvec, ηvec} = Import[
    NotebookDirectory[] <> "Data/probe_background_4d_sols_increasedAccuracy.mx"];
```

```
In[∘]:= kmin = ktabLowPrec⟦1⟧;
kmax = 10^2 kmin;
Numk = 10;
ktab = N[Table[kmin * 10^(2 j/Numk), {j, 0, Numk}], mp];

In[∘]:= ωtab = Table[0, {i, 1, Numk + 1}];
```

Initialize EQs

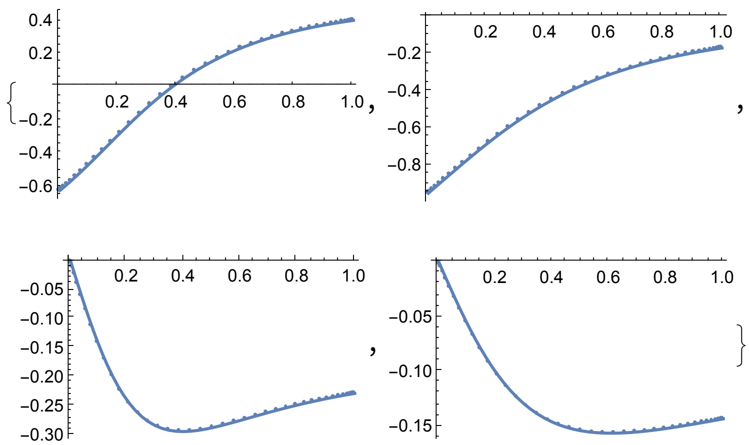
Linearize EQs

Seeds for eigenvalue problem

```
In[∘]:= (* The velocity of the superfluid sound mode is bounded above
 and below by 1 so let's try and find this eigenvalue. Furthermore,
if we are in a stable regime, Im[ω]<
 0 but Im[ω/k^2] is not too negative for the hydrodynamic modes. In Eigensystem,
eigenvalues are listed first and eigenvectors are listed second. For us,
eigs⟦wavevectorindex, 1 for eigenvalue or 2 for eigenvector, which eigenvalue⟧.
  We expect a pair of modes. *)

In[∘]:= hydroindex = Table[Position[
    Table[Abs[Re[eigsLowPrec⟦j, 1, i⟧]] / ktabLowPrec⟦j⟧ < 1 && -10 < Im[eigsLowPrec⟦j, 1, i⟧] / ktabLowPrec⟦j⟧^2 < 0,
     {i, 1, Length[eigsLowPrec⟦j, 1⟧]}], True], {j, 1, Length[ktabLowPrec]}];

In[∘]:= hydroindex
Out[∘]= {{{324}, {325}, {327}}, {{324}, {325}, {327}}}

In[∘]:= (* For the coarse grid with N=40,
we see a mode at index 324 or 325 which satisfies the hydrodynamic
 criteria for our values of k. We use one of these as our seed. *)

In[∘]:= ωseed = Table[eigsLowPrec⟦1, 1, hydroindex⟦1, 1, 1⟧⟧, {i, 1, NumGridPoints + 1}];
Export[NotebookDirectory[] <> "Data/probe_omegaseed_4d.mx", {ωseed⟦1⟧, ktab⟦1⟧}];

In[∘]:= (* Now, recall the eigenvectors have length 2*(number of fields)*
  (number of low precision grid points). The 2 is from ωδX and
  δX. We want just the fields over the grids so must partition. The
  number of fluctuating fields is the number of equations -
 1 because we have included ω'[z] as an equation. *)

In[∘]:= eigenfields = Partition[Partition[eigsLowPrec⟦1, 2, hydroindex⟦1, 1, 1⟧⟧,
      (Length[EQs] - 1) * (Length[λdataLowPrec])]⟦2⟧, Length[λdataLowPrec]];
eigenfields2 = Partition[Partition[eigsLowPrec⟦1, 2, hydroindex⟦1, 1, 1⟧⟧,
      (Length[EQs] - 1) * (Length[λdataLowPrec])]⟦1⟧, Length[λdataLowPrec]];
```

```
In[•]:= (* Check that eigenfields2 = ω*eigenfields1 *)

In[•]:= Table[Max[Chop[eigenfields2〚j〛 - ωseed〚1〛 * eigenfields〚j〛]], {j, 1, 4}]

Out[•]= {0, 0, 0, 0}

In[•]:= (* The order of the fields follows from the above
         linearization step with linFieldList. It is at, ax, σr, then σi. *)

In[•]:= (* First interpolate *)
        atint = Interpolation[
           Table[{λdataLowPrec〚i〛, eigenfields〚1, i〛}, {i, 1, Length[λdataLowPrec]}]];
        axint = Interpolation[
           Table[{λdataLowPrec〚i〛, eigenfields〚2, i〛}, {i, 1, Length[λdataLowPrec]}]];
        σrint = Interpolation[
           Table[{λdataLowPrec〚i〛, eigenfields〚3, i〛}, {i, 1, Length[λdataLowPrec]}]];
        σiint = Interpolation[
           Table[{λdataLowPrec〚i〛, eigenfields〚4, i〛}, {i, 1, Length[λdataLowPrec]}]];

In[•]:= (* Now make the seeds. Importantly,
        we have to rescale to impose our boundary condition at the horizon. *)

In[•]:= horcon = Sum[eigenfields〚j, Length[λdataLowPrec]〛, {j, 1, 4}];

In[•]:= horcon

Out[•]= -0.129100852984916706048120681023023817643980002844371657802196 +
         0.00379969064776502399501435581479130617880182712406576007484155 ⅈ

In[•]:= atseed = Table[atint[λdata〚i〛], {i, 1, NumGridPoints + 1}];
        axseed = Table[axint[λdata〚i〛], {i, 1, NumGridPoints + 1}];
        σrseed = Table[σrint[λdata〚i〛], {i, 1, NumGridPoints + 1}];
        σiseed = Table[σiint[λdata〚i〛], {i, 1, NumGridPoints + 1}];
```

In[∘]:=
```
{Show[ListLinePlot[
    Table[{λdata⟦i⟧, Re[atseed⟦i⟧]}, {i, 1, NumGridPoints + 1}]], ListPlot[
    Table[{λdataLowPrec⟦i⟧, Re[eigenfields⟦1, i⟧]}, {i, 1, Length[λdataLowPrec]}]]],
  Show[ListLinePlot[
    Table[{λdata⟦i⟧, Re[axseed⟦i⟧]}, {i, 1, NumGridPoints + 1}]], ListPlot[
    Table[{λdataLowPrec⟦i⟧, Re[eigenfields⟦2, i⟧]}, {i, 1, Length[λdataLowPrec]}]]],
  Show[ListLinePlot[
    Table[{λdata⟦i⟧, Re[σrseed⟦i⟧]}, {i, 1, NumGridPoints + 1}]], ListPlot[
    Table[{λdataLowPrec⟦i⟧, Re[eigenfields⟦3, i⟧]}, {i, 1, Length[λdataLowPrec]}]]],
  Show[ListLinePlot[
    Table[{λdata⟦i⟧, Re[σiseed⟦i⟧]}, {i, 1, NumGridPoints + 1}]], ListPlot[
    Table[{λdataLowPrec⟦i⟧, Re[eigenfields⟦4, i⟧]}, {i, 1, Length[λdataLowPrec]}]]]}
```

Out[∘]=


In[∘]:=
```
{atvec, axvec, σrvec, σivec, ωvec} = {atseed, axseed, σrseed, σiseed, ωseed};
```

In[∘]:=
```
combinedFields = Flatten[{atvec, axvec, σrvec, σivec, ωvec}];
```

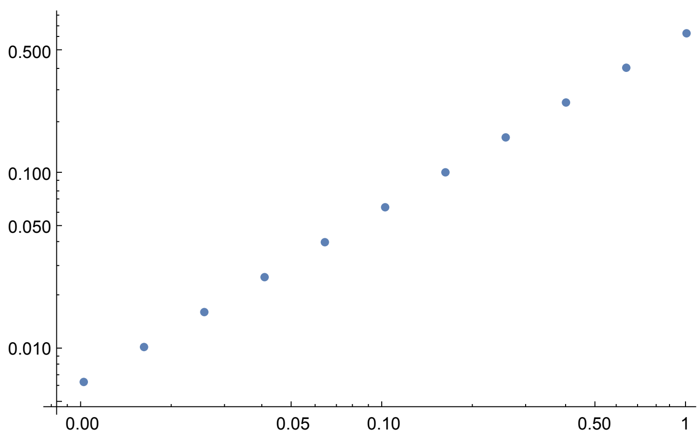
Find eigenvalues

In[∘]:=
```
nb = CreateDocument["", WindowSize → {Scaled[1 / 5], Scaled[1 / 2]}];

kcount = 1;

While[kcount ≤ Numk + 1,

  ϵp = (ErrorMax - Error) / 2;
  count = 0;

  NotebookWrite[nb,
   Cell[BoxData@RowBox[{"kcount = ", kcount, " timing = ", Timing[

         While[ϵp > Error && ϵp < ErrorMax,
```

```
μ = μtab[[μind]];
μs = μstab[[μsind]];
kk = ktab[[kcount]];


Do[coeffargs[i] = {z → λdata[[i]],
    ϕ''[z] → D2λ[[i]].ϕvec, ϕ'[z] → D1λ[[i]].ϕvec, ϕ[z] → ϕvec[[i]],
    Ax''[z] → D2λ[[i]].Axvec, Ax'[z] → D1λ[[i]].Axvec, Ax[z] → Axvec[[i]],
    η''[z] → D2λ[[i]].ηvec, η'[z] → D1λ[[i]].ηvec, η[z] → ηvec[[i]],

    at''[z] → D2λ[[i]].atvec, at'[z] → D1λ[[i]].atvec, at[z] → atvec[[i]],
    ax''[z] → D2λ[[i]].axvec, ax'[z] → D1λ[[i]].axvec, ax[z] → axvec[[i]],
    σr''[z] → D2λ[[i]].σrvec, σr'[z] → D1λ[[i]].σrvec, σr[z] → σrvec[[i]],
    σi''[z] → D2λ[[i]].σivec, σi'[z] → D1λ[[i]].σivec, σi[z] → σivec[[i]],
    ω''[z] → D2λ[[i]].ωvec, ω'[z] → D1λ[[i]].ωvec, ω[z] → ωvec[[i]]
   }, {i, 1, NumGridPoints + 1}];


(* Construct the derivative matrix *)
M = ArrayFlatten[Table[Sum[Flatten[Table[Piecewise[{{linBCcoeffs[m,
            1, n, dd] /. (coeffargs[1] /. {z → zmin}), i == 1},
          {linBCcoeffs[m, 2, n, dd] /. (coeffargs[NumGridPoints + 1] /.
              {z → zmax}), i == NumGridPoints + 1}, {linEQcoeffs[m, n,
             dd] /. coeffargs[i], i ≠ 1 && i ≠ NumGridPoints + 1}}],
        {i, 1, NumGridPoints + 1}]] × Drmatrices[[dd]], {dd, 1,
      Length[Drmatrices]}], {m, 1, Length[EQs]}, {n, 1, Length[EQs]}]];

(* Construct the value of the eoms
 given the current configuration of the fields. *)
Etot = Chop[Flatten[Parallelize[Table[Piecewise[
       {{BCs[[m, 1]] /. (coeffargs[1] /. {z → zmin}), i == 1}, {BCs[[m, 2]] /.
          (coeffargs[NumGridPoints + 1] /. {z → zmax}), i == NumGridPoints +
            1}, {EQs[[m]] /. coeffargs[i], i ≠ 1 && i ≠ NumGridPoints + 1}}],
      {m, 1, Length[EQs]}, {i, 1, NumGridPoints + 1}]]], chopmin];

(* Solve for the change in the fields *)
deltaFields = Chop[LinearSolve[M, -Etot], chopmin];

(* Compute the norm of deltaFields to keep track of convergence *)
ϵp = Norm[deltaFields, ∞];

If[NumberQ[ϵp], NotebookWrite[nb,
  Cell[BoxData@RowBox[{"ϵp= ", ϵp}], "Output"]], Break[]];
```

```
            (* If the norm is too large, introduce friction. *)
            friction = If[ϵp < 1, 1, 1 / 10];

            combinedFields = combinedFields + friction * deltaFields;
            (* partition the solution
             for the combined fields into individual fields *)
            {atvec, axvec, σrvec, σivec, ωvec} =
             Partition[combinedFields, NumGridPoints + 1];

            Clear[friction, deltaFields];

            count++;

           ];

         ]⟦1⟧, ", ♯ Iteration = ", count}], "Output"]];

   ωtab⟦kcount⟧ = ωvec⟦1⟧;
   kcount++;
  ];
 NotebookClose[nb];
```

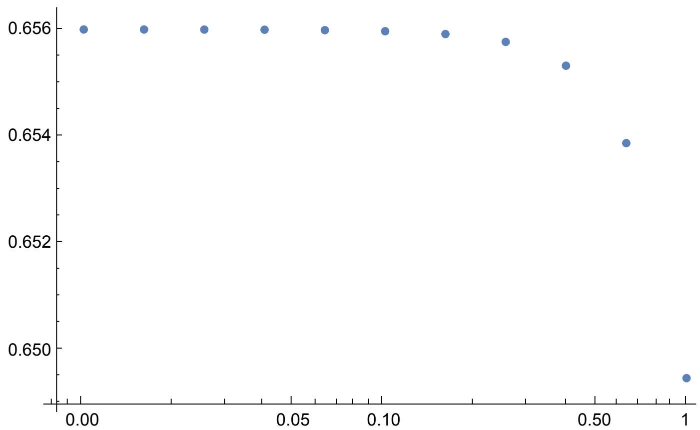
In[∘]:=
```
ListLogLogPlot[Table[{ktab⟦i⟧, Abs[Re[ωtab⟦i⟧]]}, {i, 1, Numk + 1}]]
```

Out[∘]=


In[∘]:=
```
(* Show that this is linear, i.e. a sound mode ω(k) ~ v*k *)
```

In[∘]:=
```
ListLogLogPlot[Table[{ktab⟦i⟧, Abs[Re[ωtab⟦i⟧]]/ktab⟦i⟧}, {i, 1, Numk + 1}]]
```

Out[∘]=



In[∘]:=
```
Export[
  NotebookDirectory[] <> "Data/probe_fluctuation_4d_sols_increasedAccuracy.mx",
  {μsind, μind, ktab, ωtab}];
```

5.) Matching to hydrodynamic theory

In[∘]:=
```
(* In appendix B of https://arxiv.org/pdf/2312.08243,
I write the expression for the velocity of second sound in the probe limit
 (eq. B.3). Here we will just match the result to our numerics rather than
 deriving this expression. Since we only increased precision for one point,
we will use the lower precision data for the thermodynamic derivatives. It
 is easy to change this by including more points when we increase precision.*)
```

In[∘]:=
```
Clear[μs, μ]
```

In[∘]:=
```
(* Define χξξ = μ/∂ξ(ξρs) *)
```

In[∘]:=
```
χξξ[μs_, μ_, ρs_, χnsh_] := μ/(μs * χnsh + ρs);
```

In[∘]:=
```
(* Define the velocity of second sound *)
```

In[∘]:=
```
velocityMinus[χnh_, χnn_, χξξ_] := χnh/χnn - Sqrt[χnh^2/χnn^2 + 1/(χnn χξξ)];
```

In[∘]:=
```
(* Construct thermodynamic derivatives *)
```

In[∘]:=
```
{Q, λdataLowPrec, D0λLowPrec, D1λLowPrec, D2λLowPrec,
    μtab, μstab, ϕvectabLowPrec, AxvectabLowPrec, ηvectabLowPrec} =
  Import[NotebookDirectory[] <> "Data/probe_background_4d_sols.mx"];
```

In[∘]:=
```
Nμ = Length[μtab] - 1;
Nμs = Length[μstab] - 1;
```

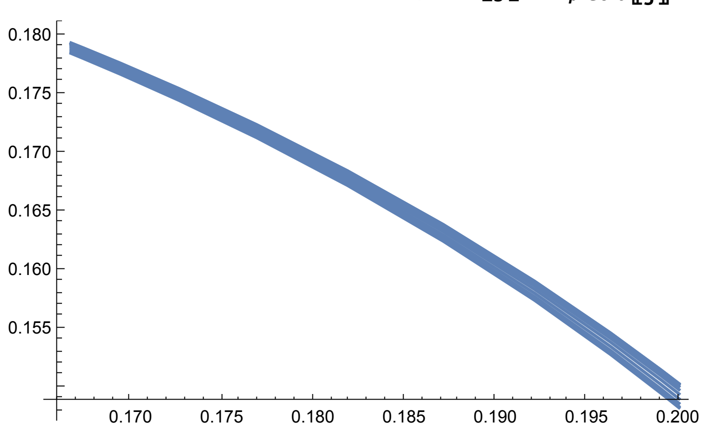
In[•]:=
```
aμ =
  N[Table[Product[If[j == k, 1, (μtab[[j]] - μtab[[k]])], {k, 1, Nμ + 1}], {j, Nμ + 1}], mp];
D1μ = Table[If[i == j, Sum[If[k == j, 0, 1/(μtab[[j]] - μtab[[k]])], {k, 1, Nμ + 1}],
    aμ[[i]]/(aμ[[j]] (μtab[[i]] - μtab[[j]]))], {i, 1, Nμ + 1}, {j, 1, Nμ + 1}];
Clear[aμ];
```

In[•]:=
```
aμs = N[Table[
    Product[If[j == k, 1, (μstab[[j]] - μstab[[k]])], {k, 1, Nμs + 1}], {j, Nμs + 1}], mp];
D1μs = Table[If[i == j, Sum[If[k == j, 0, 1/(μstab[[j]] - μstab[[k]])], {k, 1, Nμs + 1}],
    aμs[[i]]/(aμs[[j]] (μstab[[i]] - μstab[[j]]))], {i, 1, Nμs + 1}, {j, 1, Nμs + 1}];
Clear[aμs];
```

In[•]:=
```
(* Calculate the order parameter, charge density, and supercurrent density*)
Otab = Table[ηvectabLowPrec[[i, j, 1]], {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
ρtab = Table[-D1λLowPrec[[1]].ϕvectabLowPrec[[i, j]] + ϕvectabLowPrec[[i, j, 1]],
   {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
Jxtab = Table[-D1λLowPrec[[1]].AxvectabLowPrec[[i, j]], {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
ρstab = Table[μtab[[j]]/μstab[[i]] Jxtab[[i, j]], {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
```

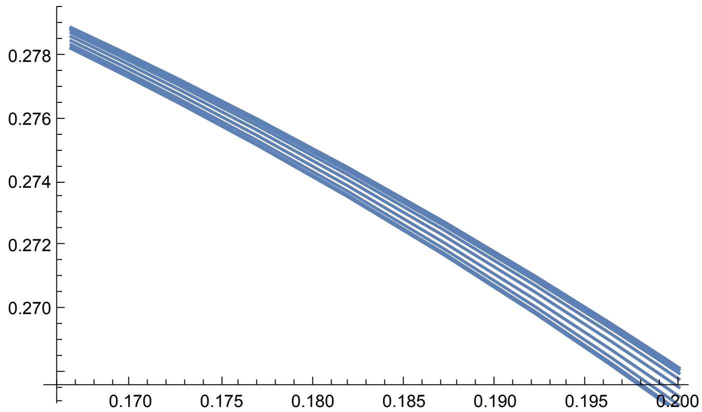
In[•]:=
```
Show[
 Table[ListLinePlot[Table[{1/μtab[[j]], Otab[[i, j]]/μtab[[j]]^2}, {j, 1, Nμ + 1}]], {i, 1, Nμs + 1}]]
```

Out[•]=



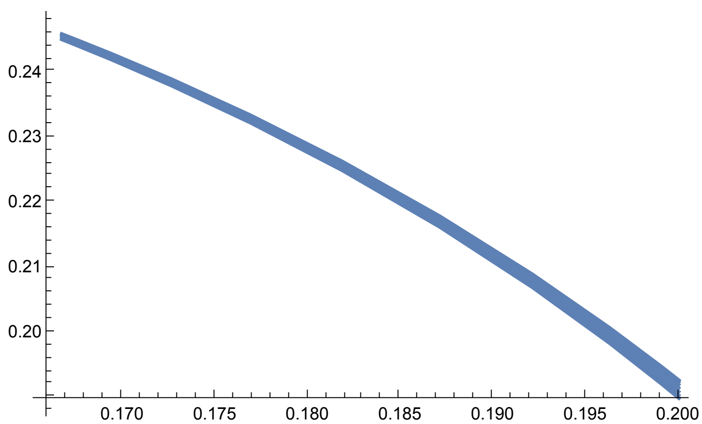
In[•]:=
```
Show[
 Table[ListLinePlot[Table[{1/μtab⟦j⟧, ρtab⟦i, j⟧/μtab⟦j⟧^2}, {j, 1, Nμ + 1}]], {i, 1, Nμs + 1}]]
```

Out[•]=



In[•]:=
```
Show[
 Table[ListLinePlot[Table[{1/μtab⟦j⟧, ρstab⟦i, j⟧/μtab⟦j⟧^2}, {j, 1, Nμ + 1}]], {i, 1, Nμs + 1}]]
```

Out[•]=



In[•]:=
```
dρdξ = D1μs.ρtab;
dρdμ = Table[D1μ⟦j⟧.ρtab⟦i⟧, {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
dρsdξ = D1μs.ρstab;
```

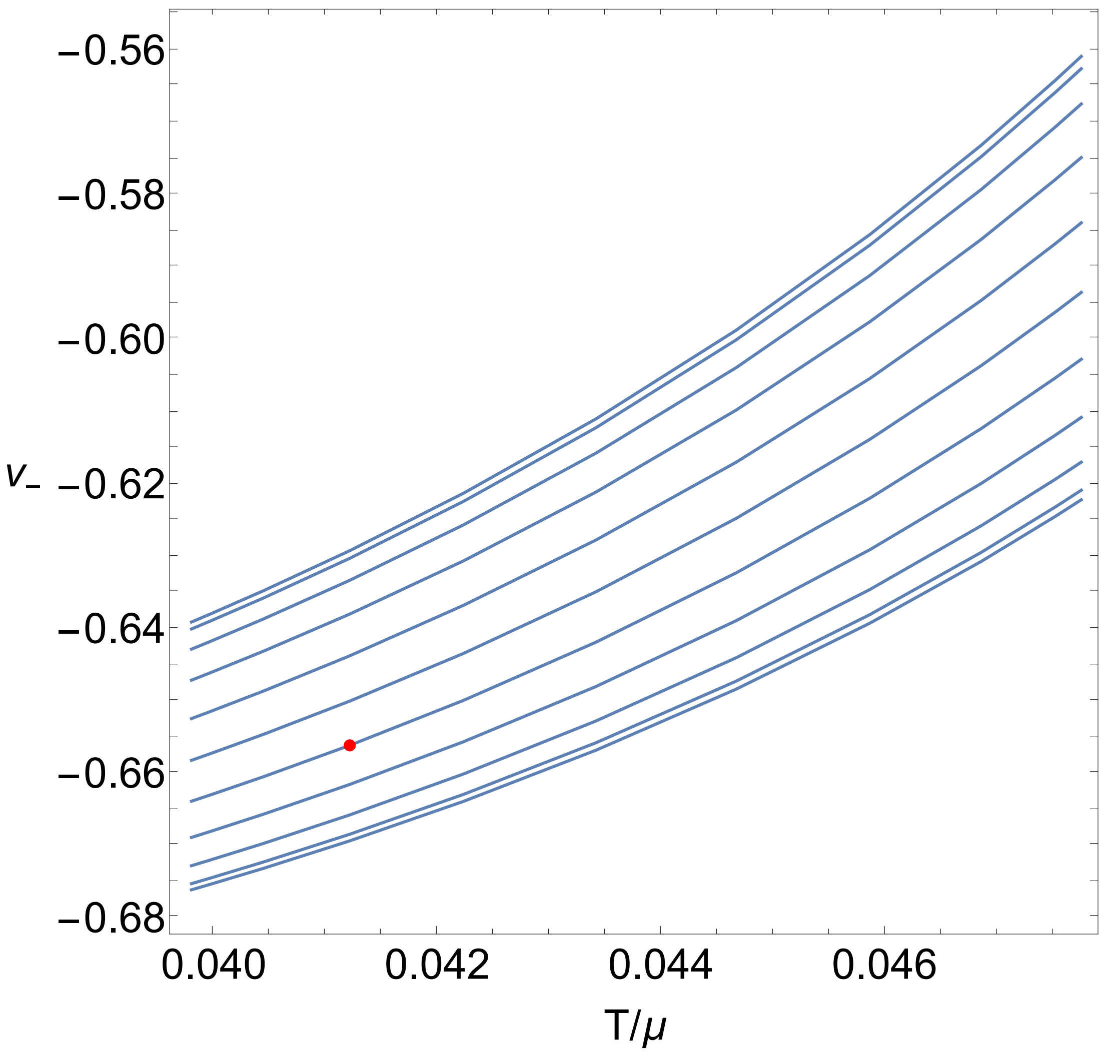
In[•]:=
```
velocityTab = Table[velocityMinus[dρdξ⟦i, j⟧, dρdμ⟦i, j⟧,
    χξξ[μstab⟦i⟧, μtab⟦j⟧, ρstab⟦i, j⟧, dρsdξ⟦i, j⟧]], {i, 1, Nμs + 1}, {j, 1, Nμ + 1}];
```

In[•]:=
```
(* Plot velocities. The red dot is where we look at the velocity*)
```

In[•]:=
```
{ωcheck, kcheck} = Import[NotebookDirectory[] <> "Data/probe_omegaseed_4d.mx"];
```

In[∘]:=
```
Show[Table[ListLinePlot[Table[{3/(4 π μtab[[j]]), velocityTab[[i, j]]}, {j, 1, Nμ + 1}],
    PlotRange → All], {i, 1, Nμs + 1}],
  ListPlot[{{3/(4 π μtab[[μind]]), Re[ωcheck / kcheck]}}, PlotStyle → Red],
  PlotRange → All, Frame → True, Axes → False, AspectRatio → 1,
  FrameLabel → {{"v_", ""}, {"T/μ", ""}}, RotateLabel → False,
  BaseStyle → {FontSize → 22}, ImageSize → Large]
```

Out[∘]=



In[∘]:=
```
velocityTab[[μsind, μind]]
```

Out[∘]= -0.655998956186536502845012096669952750391312256977564128187941

In[∘]:=
```
Re[ωcheck / kcheck]
```

Out[∘]= -0.655998634310152763995544284255483129790086710832858677329209

In[•]:=
```
Im[ωcheck / kcheck^2]
```

Out[•]=
```
-0.0284500844664581646848706393651982598002923915234086331998859
```

In[•]:=
```
(* matches to 6 significant figures. The
 discrepancy is both numerical and from finite k effects. *)
```