ARPES best fit model for the surface bands of $Sr_2 RuO_4$

Anirudh Chandrasekaran

Loughborough University
11 June 2024

Loading the Library

We first load the library by executing the following:

In[ ]:=
```
AppendTo[$Path, StringDelete[NotebookDirectory[], "/Sr2RuO4"] <> "Package"];
<< BandUtilities`
```

Interpolating between the Wannier90 files

Ingredients needed for symmetrization

In[ ]:=
```
ρ = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
(*The matrix part of the C4 rotation about one of the Ru atoms.*)
Uρ = {{0, -1, 0}, {1, 0, 0}, {0, 0, -1}};
(*Representation of C4 rotation in the d-orbital space.*)

rxy = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}}; (*x↔y reflection.*)
Urxy = {{-1, 0, 0}, {0, 1, 0}, {0, 0, -1}};
(*Representation of the x↔y reflection.*)

c0 = {1/4, 3/4, 0}; (*Location of one of the Ru atoms,
which we shall use as the centre of the C4 rotation.*)

AtomLocations =
  {{3/4, 1/4, 0}, {3/4, 1/4, 0}, {3/4, 1/4, 0}, {1/4, 3/4, 0}, {1/4, 3/4, 0}, {1/4, 3/4, 0}};
(*Location of the basis atoms in the fractional
 coordinates within the unit cell.*)

SymmetryList = {}; (*This is a multidimensional list which we shall
 construct below. Each element is a sublist corresponding to a particular
 symmetry. First element of the sublist is real space symmetry's matrix part,
second element is the translation part and the third element is
 its representation in the Hilbert space. Our point group is D4,
or D8 in some pepole's notation.*)
Do[
  AppendTo[SymmetryList,
   {MatrixPower[ρ, i], (c0 - MatrixPower[ρ, i].c0) // Simplify,
    ArrayFlatten[IdentityMatrix[2]⊗MatrixPower[Uρ, i]]}];

  AppendTo[SymmetryList,
   {MatrixPower[ρ, i].rxy, (c0 - MatrixPower[ρ, i].c0) // Simplify,
    ArrayFlatten[IdentityMatrix[2]⊗MatrixPower[Uρ, i]].
     ArrayFlatten[{{0, 1}, {1, 0}}⊗Urxy]}];
  , {i, 1, 4}];
```

Interpolation with symmetrization

We now interpolate between the Wannier90 data for 13 different rotations of the RuO octahedra: 0 ° through 12° to generate a continuously θ dependent Hamiltonian, whilst ensuring that the interpolated model has the right symmetry (for which we are feeding the list SymmetryList):

In[ ]:=
```
FileLocn = NotebookDirectory[] <> "unrenormalized2/";

W90Data = LoadInterpolateTBMs[{FileLocn <> "Sr2RuO4_0deg.dat",
     FileLocn <> "Sr2RuO4_1deg.dat", FileLocn <> "Sr2RuO4_2deg.dat",
     FileLocn <> "Sr2RuO4_3deg.dat", FileLocn <> "Sr2RuO4_4deg.dat",
     FileLocn <> "Sr2RuO4_5deg.dat", FileLocn <> "Sr2RuO4_6deg.dat",
     FileLocn <> "Sr2RuO4_7deg.dat", FileLocn <> "Sr2RuO4_8deg.dat",
     FileLocn <> "Sr2RuO4_9deg.dat", FileLocn <> "Sr2RuO4_10deg.dat",
     FileLocn <> "Sr2RuO4_11deg.dat", FileLocn <> "Sr2RuO4_12deg.dat"},
    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}, "Symmetries" → SymmetryList,
    "OrbitalCentres" → AtomLocations, "ParameterSymbol" → "θ"];

Hamltn1[k1_, k2_, θ_] = W90Data[[1]];
```

```
Invalid or no lattice vectors provided! Using unit vectors.

Tight-binding data files loaded.

Space dimension: 3

Number of orbitals: 6

Number of supercells: 49

Number of models/data-files to interpolate: 13

Number of lines in the data files:
 {741, 853, 853, 853, 845, 853, 853, 853, 853, 853, 853, 853, 853}

Interpolating between the Hamiltonians.

Valid symmetrization scheme provided! We will symmetrize the loaded Hamiltonian.

Number of supercells after symmetrization: 79
```

Phenomenological spin orbit term

We now construct the phenomenological SOC terms:

```
Lx = {{0, 0, ⅈ}, {0, 0, 0}, {-ⅈ, 0, 0}};
Ly = {{0, 0, 0}, {0, 0, ⅈ}, {0, -ⅈ, 0}};
Lz = {{0, -ⅈ, 0}, {ⅈ, 0, 0}, {0, 0, 0}};

Hupup = 1/2 Lz;
Hupdown = 1/2 (Lx + ⅈ Ly);
Hdownup = 1/2 (Lx - ⅈ Ly);
Hdowndown = -1/2 Lz;
Zero2 = ConstantArray[0, {3, 3}];

HSOC = Join[Join[Hupup, Zero2, Hupdown, Zero2, 2],
    Join[Zero2, Hupup, Zero2, Hupdown, 2], Join[Hdownup, Zero2,
     Hdowndown, Zero2, 2], Join[Zero2, Hdownup, Zero2, Hdowndown, 2]];
```

Reconstructing the full interpolated Hamiltonian

The full Hamiltonian incorporating SOC with tunable SOC strength and tunable θ. A version for the $\sqrt{2} \times \sqrt{2}$ unit cell that is tailor made for fitting to the ARPES data is also defined (since the bulk has one Ru atom per unit cell, its Brillouin zone has twice the area of the default surface band BZ, which is also 'rotated' by 45°).

```
Hamiltonian[k1_, k2_, θ_, λ_] =
  0.5 (ArrayFlatten[IdentityMatrix[2]⊗Hamltn1[k1, k2, θ]] +
     λ HSOC + ComplexExpand[ConjugateTranspose[
       ArrayFlatten[IdentityMatrix[2]⊗Hamltn1[k1, k2, θ]] + λ HSOC]]);
```

We can check the eigenvalues at a high symmetry point to see if the symmetrisation was successful, and has eliminated the weak splitting of degeneracies at the high symmetry points

```
In[ ]:= NumberForm[Sort[Eigenvalues[Hamiltonian[π, -π, 8, 0.175] // N]], 10]
```

```
Out[ ]//NumberForm=
{-0.5770901494, -0.5770901494, -0.5770901494, -0.5770901494,
 -0.0829819119, -0.0829819119, -0.0829819119, -0.0829819119,
 0.3197870614, 0.3197870614, 0.3197870614, 0.3197870614}
```

For the sake of plotting, series expansion etc, we define the following compact versions, which are also recast in terms of the bulk band Brillouin zone using the transformation $(k_x, k_y) \to (k_x - k_y, k_x + k_y)$. This is needed since the bulk band real space unit cell is smaller by a factor of 2 (in area) and lattice vectors are also $\frac{\pi}{4}$ rad rotated from the surface unit cell.

In[ ]:=
```
HamSOC[k_] := Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]], k[[4]]]

HamNoSOC[k_] := Hamltn1[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]]]

HamSOCmeV[k_] := 1000 Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]], k[[4]]]

HamNoSOCmeV[k_] := 1000 Hamltn1[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]]]
```

Evolution of the band structure with θ

We are now in a position to visualise the evolution of the band structure as θ is varied. But we first define some parameters needed for plotting, including the path in k-space

```
HSymmPts = {{{0, 0}, "Γ"}, {{π/2, π/2}, "X"}, {{0, π}, "M"}, {{0, 0}, "Γ"}};
HSymmPtsNoLabel = {{{0, 0}, ""}, {{π/2, π/2}, ""}, {{0, π}, ""}, {{0, 0}, ""}};
AngleList = {0, 3, 6, 9, 12};

OpacityList = {0.1, 0.2, 0.35, 0.5, 1};
ColourChoice = {67/255, 49/255, 12/255}; (*This for one choice of colour scheme -
 with a fixed colour and varying transparency given by the alpha channel.*)

λSOC = 0.175;
```

We now generate the band-structure data for the angles in the list using our routine which can also handle piecewise continuous k-paths (kindly have patience!! Mathematica can be a bit slow sometimes):

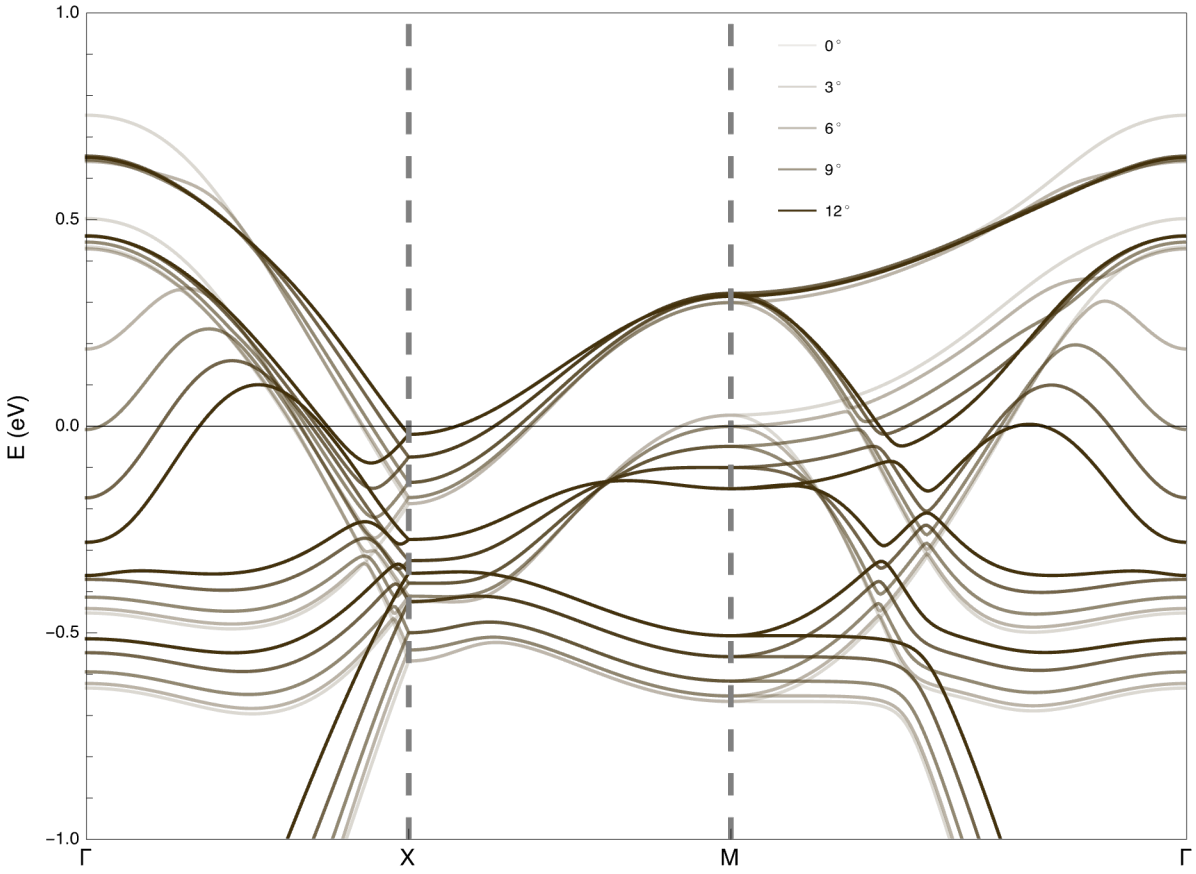
In[ ]:=
```
BandDataList = First@Last@Reap[Do[
      HamCurr[k_] := Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], angle, λSOC];

      Sow[GenerateBandStructure[HamCurr, HSymmPts, "TotalPoints" → 500]];
      , {angle, AngleList}]];
```

We will use our custom-built routine for plotting the band-structure as well:

In[ ]:=
```
BandPlotList = First@Last@Reap[Do[
    Sow[PlotBandStructure[BandDataList[[i]], {-1, 1}(*meV range*), "AspectRatio" ->
      3/4, "yLabel" -> {"E (eV)", (*Font Size*)12, (*Font Color*)Black,
       (*Font Family*)"Helvetica"}, "xLabel" -> {12, Black, "Helvetica"},
      "yTicks" -> {9, Black, "Helvetica"}, "LineThickness" -> 0.003,
      "LineColorScheme" -> {RGBColor[ColourChoice[[1]], ColourChoice[[2]],
        ColourChoice[[3]], OpacityList[[i]]]}, "DividingLines" ->
       {(*Dashing[..]*)0.02, (*Color*)Gray, (*Thickness*)0.005},
      "PlotKeyLegend" -> {Style[ToString[AngleList[[i]]] °, FontSize -> 8,
        FontFamily -> "Helvetica"], Scaled[{0.62, 1.01 - i / 20.}]}]];
  , {i, 1, Length[AngleList]}]];
```

In[ ]:=
```
SpaghettiPlot = Show[BandPlotList, ImageSize -> Full]
```

Out[ ]=



The data used to generate the figure can be saved so that it can also be plotted with other utilities like Gnuplot.

```
In[ ]:= Export[NotebookDirectory[] <> "Figures/data/Theta_0.dat", BandDataList[[1, 1]]]
        Export[NotebookDirectory[] <> "Figures/data/Theta_3.dat", BandDataList[[2, 1]]]
        Export[NotebookDirectory[] <> "Figures/data/Theta_6.dat", BandDataList[[3, 1]]]
        Export[NotebookDirectory[] <> "Figures/data/Theta_9.dat", BandDataList[[4, 1]]]
        Export[NotebookDirectory[] <> "Figures/data/Theta_12.dat", BandDataList[[5, 1]]]
Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/Figures/data/Theta_0.dat
Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/Figures/data/Theta_3.dat
Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/Figures/data/Theta_6.dat
Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/Figures/data/Theta_9.dat
Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/Figures/data/Theta_12.dat
```

We can also note the locations of the x-axis labels that will be helpful for plotting with these external programs

```
In[ ]:= BandDataList[[1, 2]] // N
Out[ ]= {{0., Γ}, {2.22144, X}, {4.44288, M}, {7.58448, Γ}}
```

Lastly, we can also note the RGB and alpha channel values that we have used above in hexadecimal base:

```
In[ ]:= Print["Red = ", BaseForm[67, 16]];
        Print["Green = ", BaseForm[49, 16]];
        Print["Blue = ", BaseForm[12, 16]];

        Do[Print["Opacity for θ = ", AngleList[[i]], "° is ",
          BaseForm[Round[255 - 255 OpacityList[[i]]], 16]], {i, Length[AngleList]}]
```

Red = 43_{16}

Green = 31_{16}

Blue = c_{16}

Opacity for θ = 0° is $e6_{16}$

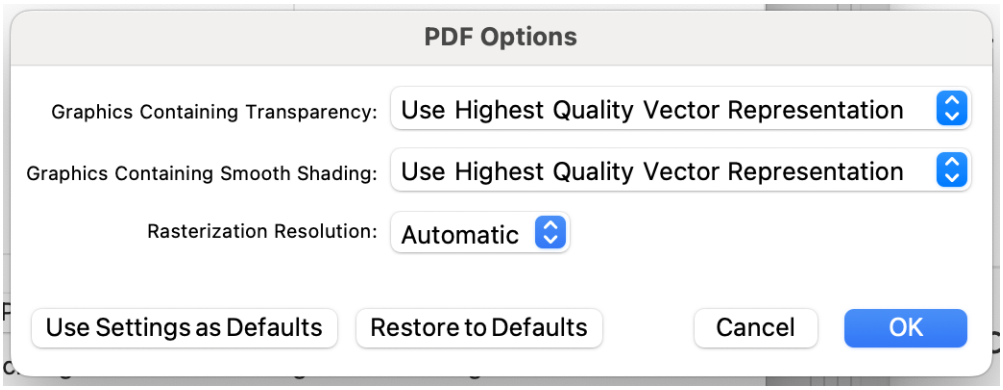
Opacity for θ = 3° is cc_{16}

Opacity for θ = 6° is $a6_{16}$

Opacity for θ = 9° is 80_{16}

Opacity for θ = 12° is 0_{16}

Nevertheless we can save SpaghettiPlot directly with somewhat decent resolution. We do right-click → Save Graphic As. In the Options set the following



Zooming into the M point

A shorter contour segment near the M point:

```
In[ ]:= HSymmPts2 =
          {{{0, π - 3.869 × 0.35}, "← Γ"}, {{0, π}, "M_y"}, {{3.869 × 0.35, π}, "X →"}};
        ColourChoice2 = {45/255, 38/255, 87/255};
```

Generating the band-structure data:

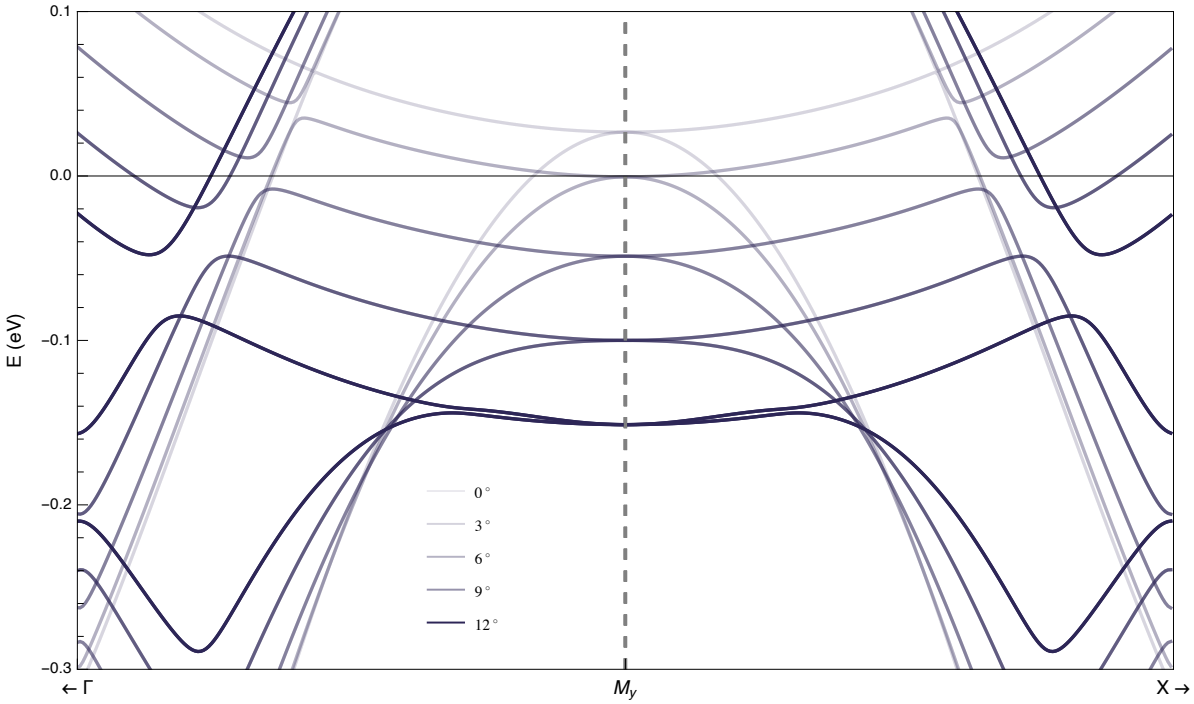
```
In[ ]:= BandDataList2 = First@Last@Reap[Do[
          HamCurr[k_] := Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], angle, λSOC];

          Sow[GenerateBandStructure[HamCurr, HSymmPts2, "TotalPoints" → 400]];
          , {angle, AngleList}]];
```

Plotting this:

```
In[ ]:= BandPlotList2 = First@Last@Reap[Do[
          Sow[PlotBandStructure[BandDataList2[[i]], {-0.3, 0.1}, "AspectRatio" → 3/5,
            "yLabel" → {"E (eV)", (*Font Size*)10, (*Font Color*)Black,
              (*Font Family*)"Helvetica"}, "xLabel" → {10, Black, "Helvetica"},
            "yTicks" → {8, Black, "Helvetica"}, "LineThickness" → 0.003,
            "LineColorScheme" → {RGBColor[ColourChoice2[[1]], ColourChoice2[[2]],
               ColourChoice2[[3]], OpacityList[[i]]]}, "DividingLines" →
             {(*Dashing[..]*)0.01, (*Color*)Gray, (*Thickness*)0.003},
            "PlotKeyLegend" → {Style[ToString[AngleList[[i]]]°, FontSize → 8,
               FontFamily → "Times"], Scaled[{0.31, 0.32 - i / 20.}]}]];
          , {i, 1, Length[AngleList]}]];
```

In[ ]:=
```
SpaghettiPlot2 = Show[BandPlotList2, ImageSize → Full]
```

Out[ ]=



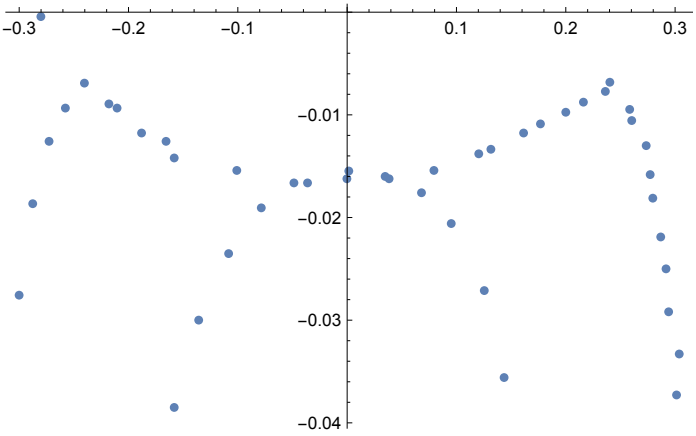
Loading and cleaning ARPES data

Loading the data

Finally we load the ARPES data:

In[ ]:=
```
EData = Reverse[#] & /@ Import[NotebookDirectory[] <> "/Surface_data_copy.txt",
    "Table"(*,"FieldSeparators"→" ","LineSeparators"→","*)];
(*Reverse[#]&/@EData*)
ListPlot[EData]
```

Out[ ]=



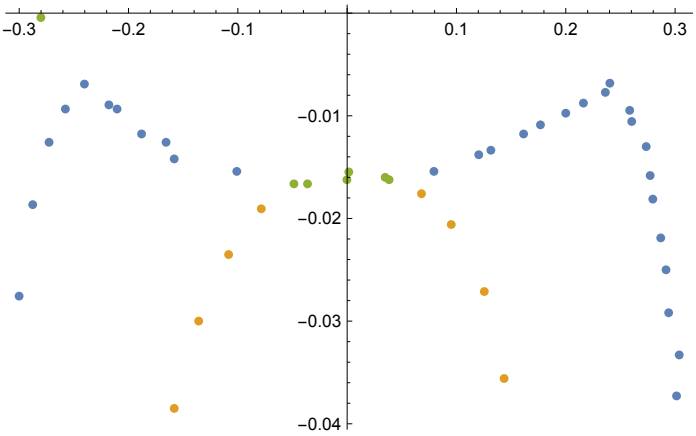
Cleaning the data

We have to do much to clean up the data and recast it in a format suitable for comparison to the-

ory. We begin by attempting to decompose the data into distinct bands:

In[ ]:=
```
Band1 = {};
Band2 = {};
BandBad = {};
Do[
 If[Abs[kData[[1]]] > 0.16 && kData[[2]] < -0.004,
  AppendTo[Band1, kData]
  ,
  If[-0.16 < kData[[1]] < -0.06 || 0.16 > kData[[1]] > 0.04,
   If[kData[[2]] < -0.017,
    AppendTo[Band2, kData],
    AppendTo[Band1, kData]
   ]
   ,
   AppendTo[BandBad, kData];
  ]
 ]
 , {kData, EData}]
ListPlot[{Band1, Band2, BandBad}]
```

Out[ ]=



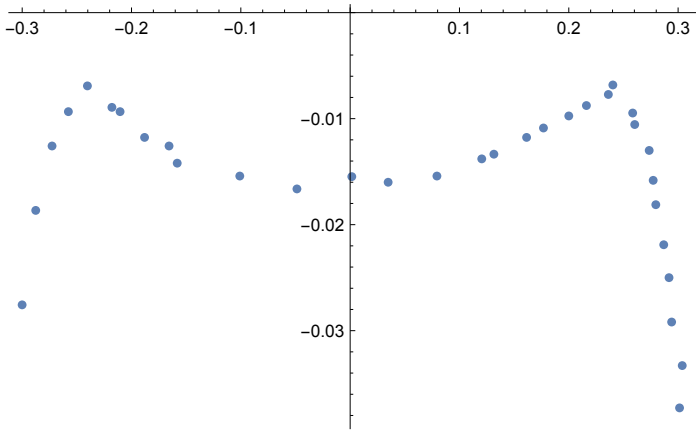
Since it is not immediately clear if the green dots belong to upper or lower band, we put them into a separate list, before cleaning it up.

In[ ]:=
```
BandBad = SortBy[BandBad, First];
```

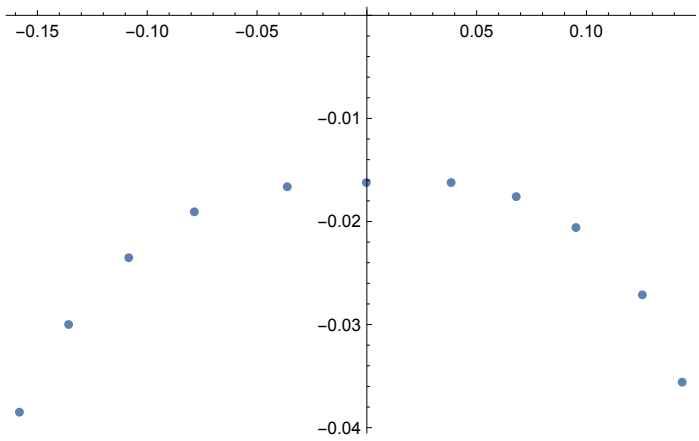
Decomposing into distinct bands

In[ ]:=
```
BandUp = SortBy[Join[Band1, {BandBad[[2]], BandBad[[5]], BandBad[[6]]}], First];
BandDown = SortBy[Join[Band2, BandBad[[3 ;; 4]], {BandBad[[7]]}], First];
ListPlot[BandUp]
ListPlot[BandDown]
```

Out[ ]=



Out[ ]=



Recasting ARPES data with full **k**-vectors (for comparison with theory)

Thus, we have decomposed the data into two distinct bands (up and down bands). We have to store this data into a list with the actual **k** vectors rather than **k**-lengths

In[ ]:=
```
BandPathData = {};
Do[
 CurData = {};
 If[kData[[1]] < 0,
  AppendTo[CurData, {0, π + 3.869 kData[[1]]}];
  AppendTo[CurData, {kData[[2]]}];
  AppendTo[CurData, {7}];
  AppendTo[BandPathData, CurData];
  ,
  AppendTo[CurData, {3.869 kData[[1]], π}];
  AppendTo[CurData, {kData[[2]]}];
  AppendTo[CurData, {7}];
  AppendTo[BandPathData, CurData];

 ];
 , {kData, Band1(*Change to BandUp if needed*)}]

Do[
 CurData = {};
 If[kData[[1]] < 0,
  AppendTo[CurData, {0, π + 3.869 kData[[1]]}];
  AppendTo[CurData, {kData[[2]]}];
  AppendTo[CurData, {6}];
  AppendTo[BandPathData, CurData];
  ,
  AppendTo[CurData, {3.869 kData[[1]], π}];
  AppendTo[CurData, {kData[[2]]}];
  AppendTo[CurData, {6}];
  AppendTo[BandPathData, CurData];
 ];
 , {kData, Band2(*Change to BandDown if necessary*)}]
BandPathData;
```

A function to tune and fit the band structure **(Evaluate before proceeding further!)**

Fitting the ARPES data

In[ ]:=
```
TuneBandStructure[HamSOC, 2, 2, {{0, 12}, {0.01, 0.4}}, {130, 50}, BandPathData]
```

```
Correct data format!

Max number of bands = 1

Done!!
```

We will still have to compile and run the cpp program

```
In[ ]:= Print["Run the following commands on Linux terminal in sequence:\n",
          "cd " <> NotebookDirectory[] <> "cpp/ \n",
          "g++ Tune.cpp -o Tune -llapack -lblas -fopenmp \n", "./Tune"];
```

```
Run the following commands on Linux terminal in sequence:
cd /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/cpp/
g++ Tune.cpp -o Tune -llapack -lblas -fopenmp
./Tune
```

If you use a Mac, things can get a bit complicated since the native support for OpenMP was disabled some time back. I used XCode and then Homebrew to install and link it. In that situation, the library and include directories have to be found and included. For example, I used the following

```
In[ ]:= Print["Instead run the following commands on a Mac terminal:\n",
          "cd " <> NotebookDirectory[] <> "cpp/ \n",
          "clang++ Tune.cpp -o Tune -llapack -lblas
            -Xclang -fopenmp -lomp -L/opt/homebrew/opt/libomp/lib
            -I/opt/homebrew/opt/libomp/include \n", "./Tune"];
```

```
Instead run the following commands on a Mac terminal:
cd /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/cpp/
clang++ Tune.cpp -o Tune -llapack -lblas -Xclang -fopenmp -lomp
   -L/opt/homebrew/opt/libomp/lib -I/opt/homebrew/opt/libomp/include
./Tune
```

After running the program we can read its output

```
In[ ]:= {θFit, λFit, ZFit, E0Fit, ErrFit} =
          Import[NotebookDirectory[] <> "cpp/TuningSolution.dat"][[1]];
        Print["Best fit parameters: θ = ", θFit, ", λSOC = ", λFit,
          ", Z = ", ZFit, ", EF = ", E0Fit, ", % Error = ", 100 ErrFit];
```

```
Best fit parameters: θ = 8.03077, λSOC = 0.1738
 , Z = 0.244404, EF = -0.00262535 , % Error = 0.0139292
```

Using this, we define a number of Hamiltonians to be used for different purposes

```
In[ ]:= HamFit[k_] := ZFit Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], θFit, λFit] -
          DiagonalMatrix[ConstantArray[E0Fit, 12]]
        HamFitNoSOC[k_] := ZFit Hamltn1[k[[1]] - k[[2]], k[[1]] + k[[2]], θFit] -
          DiagonalMatrix[ConstantArray[E0Fit, 6]]
        HamTuneNoSOC[k_] := ZFit Hamltn1[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]]] -
          DiagonalMatrix[ConstantArray[E0Fit, 6]]
        HamTuneNoSOCmeV[k_] := 1000 ZFit Hamltn1[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]]] -
          1000 DiagonalMatrix[ConstantArray[E0Fit, 6]]
        HamTuneSOCmeV[k_] := 1000 ZFit Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]], λFit] -
          1000 DiagonalMatrix[ConstantArray[E0Fit, 12]]
        HamFullTuneSOCmeV[k_] :=
         1000 ZFit Hamiltonian[k[[1]] - k[[2]], k[[1]] + k[[2]], k[[3]], k[[4]]] -
          1000 DiagonalMatrix[ConstantArray[E0Fit, 12]]
```

Comparing ARPES bands with the bands from the best -

fit Hamiltonian - first pass

Generating the band-structure data to plot

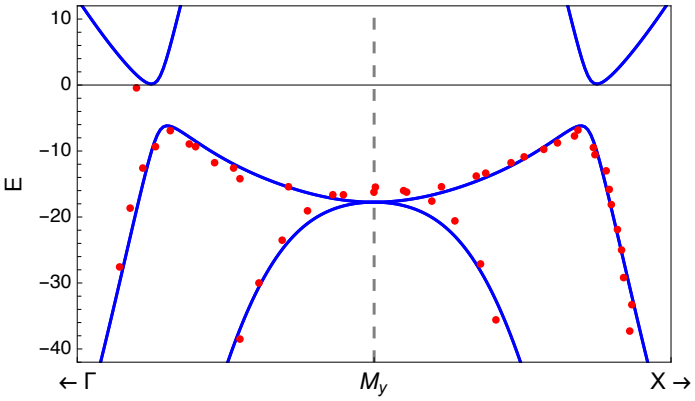
```
In[ ]:= HamFitmeV[k_] := 1000 HamFit[k];
        Bnd1 = GenerateBandStructure[HamFitmeV, HSymmPts2];
```

Plotting the best fit band-structure and ARPES data together for comparison

Finally we plot the band of the theory fit Hamiltonian along with the experimental data in meV:

```
In[ ]:= plt1 = PlotBandStructure[Bnd1, (*{-0.042,0.012}*){-42, 12}, "AspectRatio" → 3/5];
        plt2 = ListPlot[({3.869 (#[[1]] + 0.35), 1000 #[[2]]}) & /@ EData, PlotStyle → Red];
        Show[plt1, plt2]
```

Out[ ]=



Writing the interpolated Wannier90 data

Time to write the interpolated Hamiltonian into Wannier90 format, with and without the SOC term:

```
In[ ]:= WriteTBM[NotebookDirectory[] <> "output_models/Sr2RuO4_8p03degrees_SOC_hr.dat",
          W90Data[[2]], "ParameterValue" → θFit,
          "ParameterSymbol" → "θ", "BandRenormalization" → ZFit,
          "FermiLevel" → E0Fit, "SpinOrbitMatrix" → λFit HSOC];

        WriteTBM[
          NotebookDirectory[] <> "output_models/Sr2RuO4_8p03degrees_noSOC_hr.dat",
          W90Data[[2]], "ParameterValue" → θFit, "ParameterSymbol" → "θ",
          "BandRenormalization" → ZFit, "FermiLevel" → E0Fit];

        Valid SOC matrix provided. Doubling
          the number of orbitals and writing the Wannier90 data.

        Invalid or no SOC matrix provided! Writing an SOC-free Hamiltonian.
```

Sanity Checks

We load the saved Wannier tight binding model and check if it reproduces the ARPES comparison figure

In[ ]:=
```
Hamiltonian2[k1_, k2_] =
  LoadTBM[NotebookDirectory[] <> "output_models/Sr2RuO4_8p03degrees_SOC_hr.dat",
    "FileType" → "Wannier90"][[1]];
```

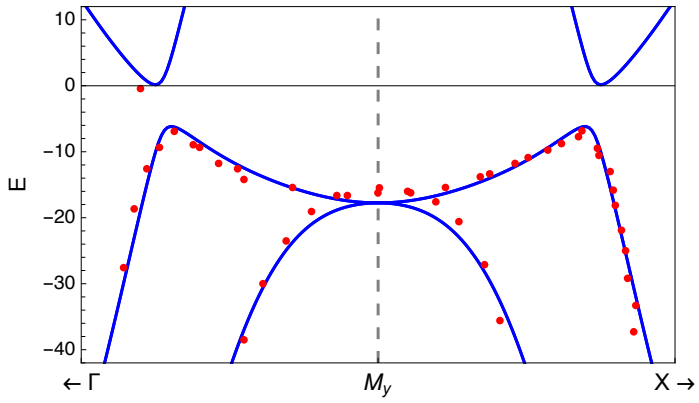
```
Invalid or no lattice vectors provided! Using unit vectors.
Wannier data begins at line 10.
Tight-binding data file loaded.
Space dimension: 3
Number of orbitals: 12
Number of supercells: 79
First row of data: {-4, -3, 0, 1, 1, 0., 0.}
Last row of data: {4, 3, 0, 12, 12, 0., 0.}
Number of entries: 11376
Loading Hamiltonian.
No valid symmetrization scheme
  provided! We will not symmetrize the loaded Hamiltonian.
```

Using this we define another Hamiltonian to be used exclusively for plotting the band

In[ ]:=
```
HamFitmeV2[k_] := 1000 Hamiltonian2[k[[1]] - k[[2]], k[[1]] + k[[2]]];
Bnd2 = GenerateBandStructure[HamFitmeV2, HSymmPts2];
```

In[ ]:=
```
plt3 = PlotBandStructure[Bnd2, (*{-0.042,0.012}*){-42, 12}, "AspectRatio" → 3/5];
Show[plt3, plt2]
```

Out[ ]=



Thus we have verified that it neatly reproduces the original comparison.

The Fermi surface

We will use brute force to calculate the Fermi surface. The functions to do this also need significant improvements to improve their useability and have therefore not been included in the library

Extra function **(execute this before proceeding further)**

Generating the Fermi surface

Firstly we write the Hamiltonian to a C++ file

In[ ]:=
```
WriteHamiltonianCPP[HamFit, 2]
```

```
Invalid or no file path provided! Reverting
  to the default name Hamltn.cpp. File shall be stored in the
  directory cpp located in the local directory of the notebook.

Hamiltonian successfully written to Hamltn.cpp
 . Please include both Hamltn.cpp and ancillaries.cpp in your code.
```

Before running the routine to generate the Fermi surface, compile FermiSurface.cpp with flags -llapack and -lblas. The file is located in the cpp folder.

In[ ]:=
```
GenerateFermiSurfaceData[0., 2, {{-π, π}, {-π, π}}]
```

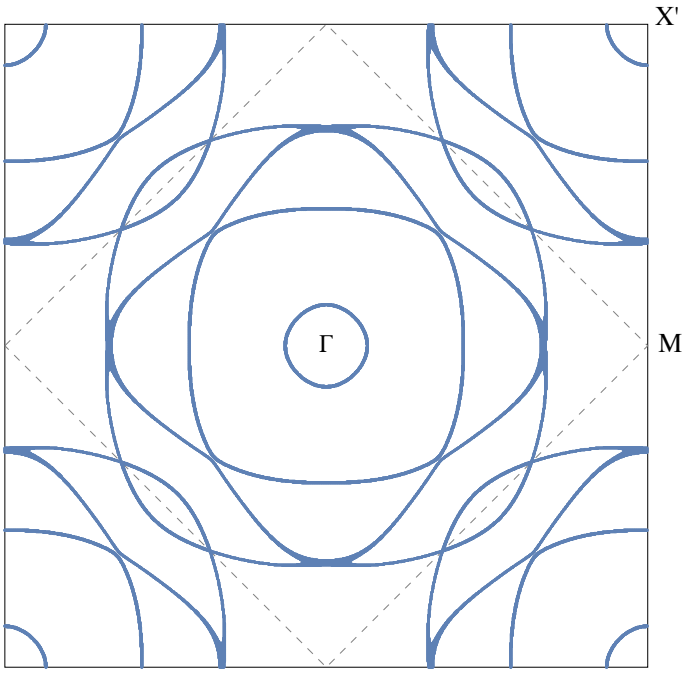
```
Invalid grid or no grid provided! Using a 1000x1000 grid of k points.

Now going to run (cd
  /Users/phac/Research/GitHub/BandUtilities/Sr2RuO4/cpp/ ; ./FermiSurface)
```

Out[ ]=
```
InputStream[ Name: FS.dat
             Unique ID: 4 ]
```

Plotting the Fermi surface

In[ ]:=
```
file3 = NotebookDirectory[] <> "cpp/FS.dat";
(*Close[strm3]*)
strm3 = OpenRead[file3];

F3 = Read[strm3];
Close[strm3];
FermiList3 = First@
    Last@Reap[Do[If[Length[band] ≠ 0, Sow[ListPlot[band[[All, 1]], AspectRatio → 1,
          Axes → False, Ticks → None, Frame → True, FrameTicks → None,
          PlotRange → {{-π, π}, {-π, π}}]]], {band, F3}]];
pltFS = Show[Graphics[
    Text[Style["Γ", FontSize → 14, FontFamily → "Times"], {0, 0}]], Graphics[
    Text[Style[" X'", FontSize → 14, FontFamily → "Times"], {1.05 π, 1.02 π}]],
  Graphics[Text[Style["  M", FontSize → 14, FontFamily → "Times"], {1.05 π, 0}]],
  Graphics[{EdgeForm[{Black}], FaceForm[], Rectangle[{-π, -π}, {π, π}]}],
  Graphics[{EdgeForm[{Gray, Dashed}], FaceForm[],
    Polygon[{{-π, 0}, {0, π}, {π, 0}, {0, -π}}]}], FermiList3]

Export[NotebookDirectory[] <> "Fermi_Surface.pdf", Show[pltFS, ImageSize → 7 cm]]
```

Out[ ]=

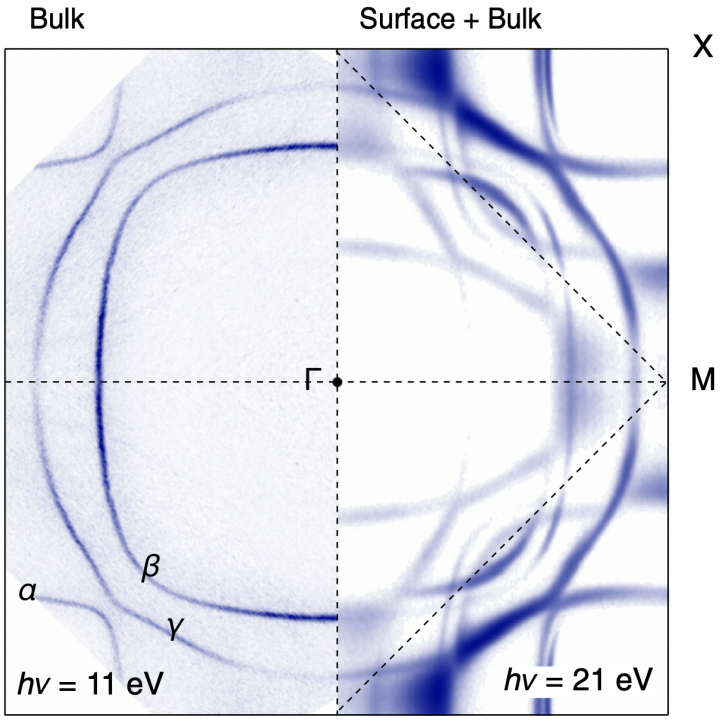


Out[ ]= /Users/phac/Research/GitHub/BandUtilities/Sr2Ru04/Fermi_Surface.pdf

Loading the experimental Fermi surface below (sourced from Fig 7 of Phys. Rev. X 9, 021048, https://journals.aps.org/prx/abstract/10.1103/PhysRevX.9.021048), for comparison:

In[ ]:=
```
Import[NotebookDirectory[] <> "Fermi_Surface_expt.png"]
```

Out[ ]=



For a more careful comparison check FIG. 3 of our paper: https://arxiv.org/pdf/2310.15331