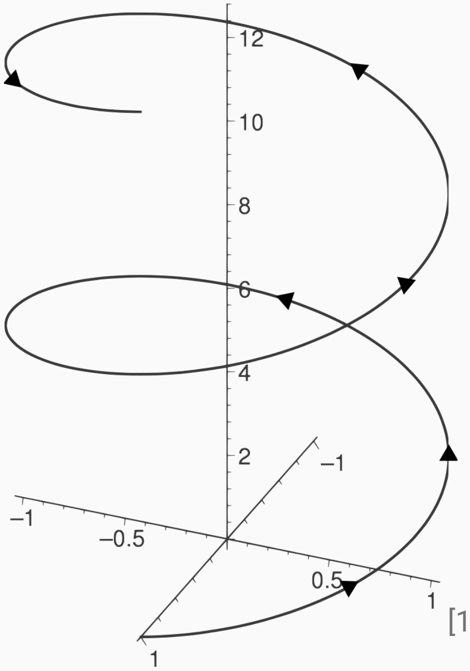
Develop of Clad Tutorials for CMS/HEP

Austėja Jurgaitytė
Mentor: David Lange

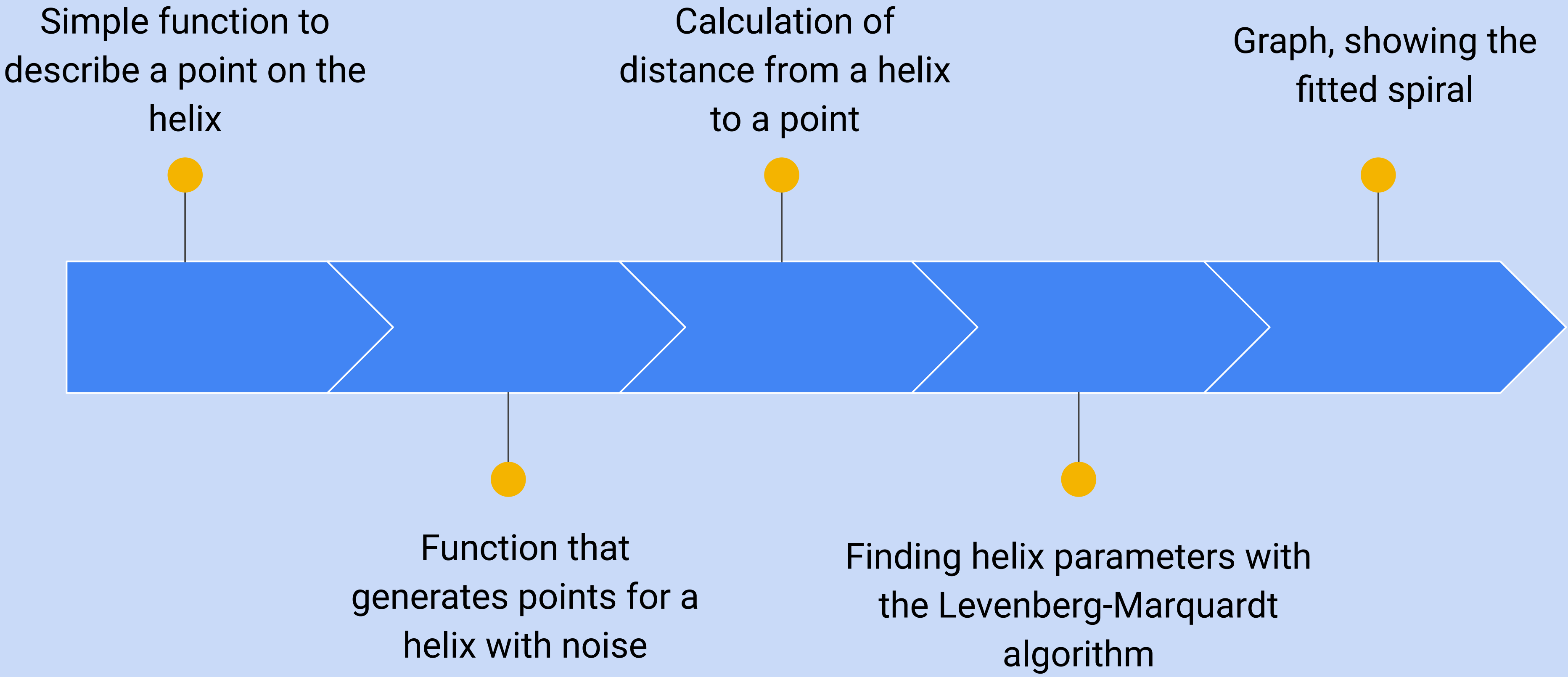
Project goals

- Creating a Clad based demonstration of finding the best fit helix parameters given a set of data points.
- Contribute to Clad code fixing the missing functionalities that we find along the way



[1]

The plan for the main tutorial



Simple function to describe a point on the helix

Function that generates points for a helix with noise

Calculation of distance from a helix to a point

Finding helix parameters with the Levenberg-Marquardt algorithm

Graph, showing the fitted spiral

Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm combines two optimization methods: gradient descent and Gauss-Newton.

Its behaviour changes based on how close the current coefficients are to the optimal value.

The equation that dictates how to update the parameters in the Levenberg-Marquardt algorithm is this:

$$(J^TWJ + \lambda I)\ h_{lm} = J^T\ W\ (y - \hat{y})$$

Distance to point calculations

- To find the closest distance of a point to a helix, we do some scaling so that our helix is now defined by $(\cos t, \sin t, \hbar t)$.
- For a given point P(i,j,k), let Q be the closest point on the helix. The line segment connecting P and Q must be perpendicular to the helix's tangent line at Q, which is just $(-\sin t, \cos t, \hbar)$:

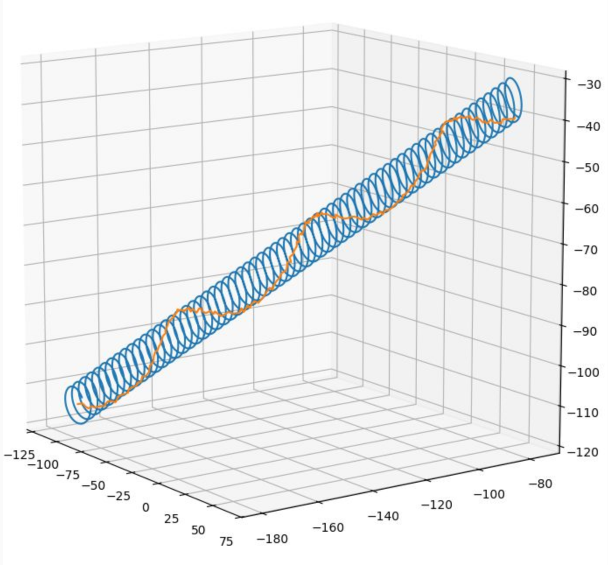
$$-(\cos t - \mathrm{i})\sin t + (\sin t - \mathrm{j})\cos t + (\hbar t - \mathrm{k})\hbar = 0$$

- This simplifies to $A\sin(t+B)+Ct+D=0$ for some constants A,B,C,D.
- To find the solution, I perform a binary search

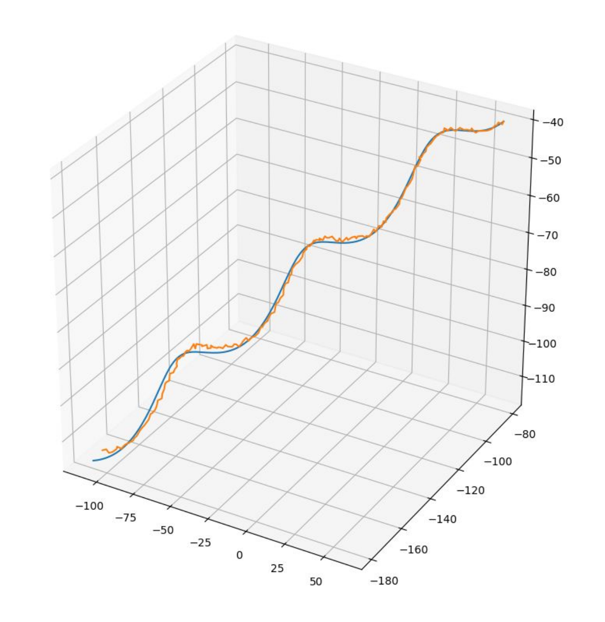
The Jacobian

- There were some problems deciding which Clad function to use.
- Clad’s execute method doesn’t seem to support arguments that have length specified as a variable, so I decided not to use clad::jacobian.
- I decided to write my own function, using clad::differentiate

Graphs



−30
−40
−50
−60
−70
−80
−90
−100
−110
−120
−125
−100
−75
−50
−25
0
25
50
75
−180
−160
−140
−120
−100
−80



−40
−50
−60
−70
−80
−90
−100
−110
−100
−75
−50
−25
0
25
50
−180
−160
−140
−120
−100
−80

Gradient Descent

- Perhaps a better way to showcase Clad (but not necessarily a better way to approximate a helix)
- the implementation found in fitter.h gets stuck in a local minimum that is very far off from the actual expected results.

Problems with tutorial format

- I would prefer a Jupyter notebook-style tutorial
- My Jupyter has connection errors when using xeus-cling kernels (HTTP 404: Not Found (Kernel does not exist))
- Currently my tutorial consists of the code in my Github repository and a pdf.

What I learned

- I gained knowledge about automatic differentiation and Clad.
- Learned more about C++.
- Refreshed my knowledge about various fitting methods.
- Got experience working with a new mentor.
- Got a taste of what it’s like to work with a team.

References

[1] Helix Wikipedia page, https://en.wikipedia.org/wiki/Helix

[2] Clad GitHub, [https://github.com/vgvassilev/clad]

[3] The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems, https://people.duke.edu/~hpgavin/lm.pdf

[4] Shortest distance between a point and a helix, https://math.stackexchange.com/questions/13341/shortest-distance-between-a-point-and-a-helix

Project GitHub [https://github.com/ZeptoStarling/Clad-IRIS-HEP-2024]