Diff:

Differences between given skeleton and solution
In order to make the sample solution easier to understand, the differences between it and the given skeleton source code were highlighted with the help of the program `diff`.

Legend:

- Gray: unchanged text (only excerpts).
- Green: new lines
- Yellow: changed lines
- Red: deleted lines

Note: Files not listed have not been changed.

This document was created with the help of diff2html erstellt.

diff -u ../course04-2d-visualization/exercise/code/01_plot.py ../course04-2d-visualization/exercise/solution/01_plot.py

../course04-2d-visualization/exercise/code/01_plot.py

```
⋮
zz0 = measurment_data[:, 0] # first column of measurement data -> initial value of the simulation

# Create empty list for collecting intermediate optimization results:
resList = XXX


# this is copied from exercise 03.2 (+ one line added)
⋮
   # Exercise 03.1:
   # That single line is new:
   # save the current simulation result (state array) in the prepared list
   XXX.append(XXX)



⋮


## After successful saving, insert a 0 here:
if 1:
   # Optimization - result are the two parameters m2 and l

   # intially guessed values for optimization
⋮
fig = plt.figure(figsize=(300*mm, 170*mm))

# subplot for unoptimized curves
ax1 = fig.add_subplot(XXX, XXX, XXX)
ax1.plot(tt, XXX[XXX, :], label="measurement")
ax1.plot(tt, resList[0][XXX, XXX], label="model (initial guess)")
ax1.legend()
ax1.grid()
ax1.set_ylabel("distance $x ~[m]$")


# subplot for optimization result
ax2 = XXX.add_XXX(XXX, XXX, XXX)
ax2.plot(XXX, XXX, lw=3, label="measurement")
# ...


# subplot for remaining error of x position
# ...


plt.savefig("fig1.pdf", bbox_inches="tight")
# plt.show()
⋮
```

../course04-2d-visualization/exercise/solution/01_plot.py

```
⋮
zz0 = measurment_data[:, 0] # first column of measurement data -> initial value of the simulation

# Create empty list for collecting intermediate optimization results:
resList = []


# this is copied from exercise 03.2 (+ one line added)
⋮
   # Exercise 03.1:
   # That single line is new:
   # save the current simulation result (state array) in the prepared list
   resList.append(zz_res)



⋮


## After successful saving, insert a 0 here:
if 0:
   # Optimization - result are the two parameters m2 and l

   # intially guessed values for optimization
⋮
fig = plt.figure(figsize=(300*mm, 170*mm))

# subplot for unoptimized curves
ax1 = fig.add_subplot(3, 1, 1)
ax1.plot(tt, measurment_data[0, :], label="measurement")
ax1.plot(tt, resList[0][0, :], label="model (initial guess)")
ax1.legend()
ax1.grid()
ax1.set_ylabel("distance $x ~[m]$")


# subplot for optimization result
ax2 = fig.add_subplot(3, 1, 2)
ax2.plot(tt, measurment_data[0, :], lw=3, label="measurement")
ax2.plot(tt, resList[-1][0, :], label="model (optimized parameters)")
ax2.legend()
ax2.grid()
ax2.set_ylabel("distance $x ~[m]$")


# subplot for remaining error of x position
ax3 = fig.add_subplot(3, 1, 3)
ax3.plot(tt, measurment_data[0, :]-resList[-1][0, :])
ax3.grid()
ax3.set_xlabel("time $t ~[s]$")
ax3.set_ylabel("error $\epsilon$ ~[m]")


plt.savefig("fig1.pdf", bbox_inches="tight")
# plt.show()
⋮
```

```python
    colStep = 0.9 / len(resList)
    gray_value = str(0.01 + (0.9 - i*colStep))

    plt.plot(XXX, XXX[XXX][XXX, XXX], color=gray_value )

# Plotting the measurement and the simulation
plt.plot(XXX, XXX, color="#3366FF", lw=4, label="measurement")
plt.plot(XXX, XXX[-1][XXX, XXX], color="#FF9900", ls="--", lw=2, label="model (optimized parameters)")

# grid
# legend
# x label
# y label


# here we use png because due to the many individual lines the pdf would be quite big and slow
plt.savefig("fig2.png", bbox_inches="tight")
```

```python
    colStep = 0.9 / len(resList)
    gray_value = str(0.01 + (0.9 - i*colStep))

    plt.plot(tt, resList[i][0, :], color=gray_value )

# Plotting the measurement and the simulation
plt.plot(tt, measurment_data[0, :], color="#3366FF", lw=4, label="measurement")
plt.plot(tt, resList[-1][0, :], color="#FF9900", ls="--", lw=2, label="model (optimized parameters)")
plt.grid()
plt.legend()
plt.xlabel("time $t ~[s]$")
plt.ylabel("distance $x ~[m]$")

# here we use png because due to the many individual lines the pdf would be quite big and slow
plt.savefig("fig2.png", bbox_inches="tight")
```

⋮

```python

# overview on color maps:
# https://matplotlib.org/examples/color/colormaps_reference.html
ax.plot_surface(XXX, rstride=1, cstride=1, cmap=cm.XXX)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
```

```python

# overview on color maps:
# https://matplotlib.org/examples/color/colormaps_reference.html
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.viridis)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
```

```
Nur in ../course04-2d-visualization/exercise/solution/: fig1.pdf.
Nur in ../course04-2d-visualization/exercise/solution/: fig2.png.
Nur in ../course04-2d-visualization/exercise/solution/: fig3.png.
Nur in ../course04-2d-visualization/exercise/solution/: __pycache__.
Nur in ../course04-2d-visualization/exercise/solution/: res_list.pcl.
```