# Lecture 13: inverse kinematics II

Prof. Katie Driggs-Campbell

March 24, 2026

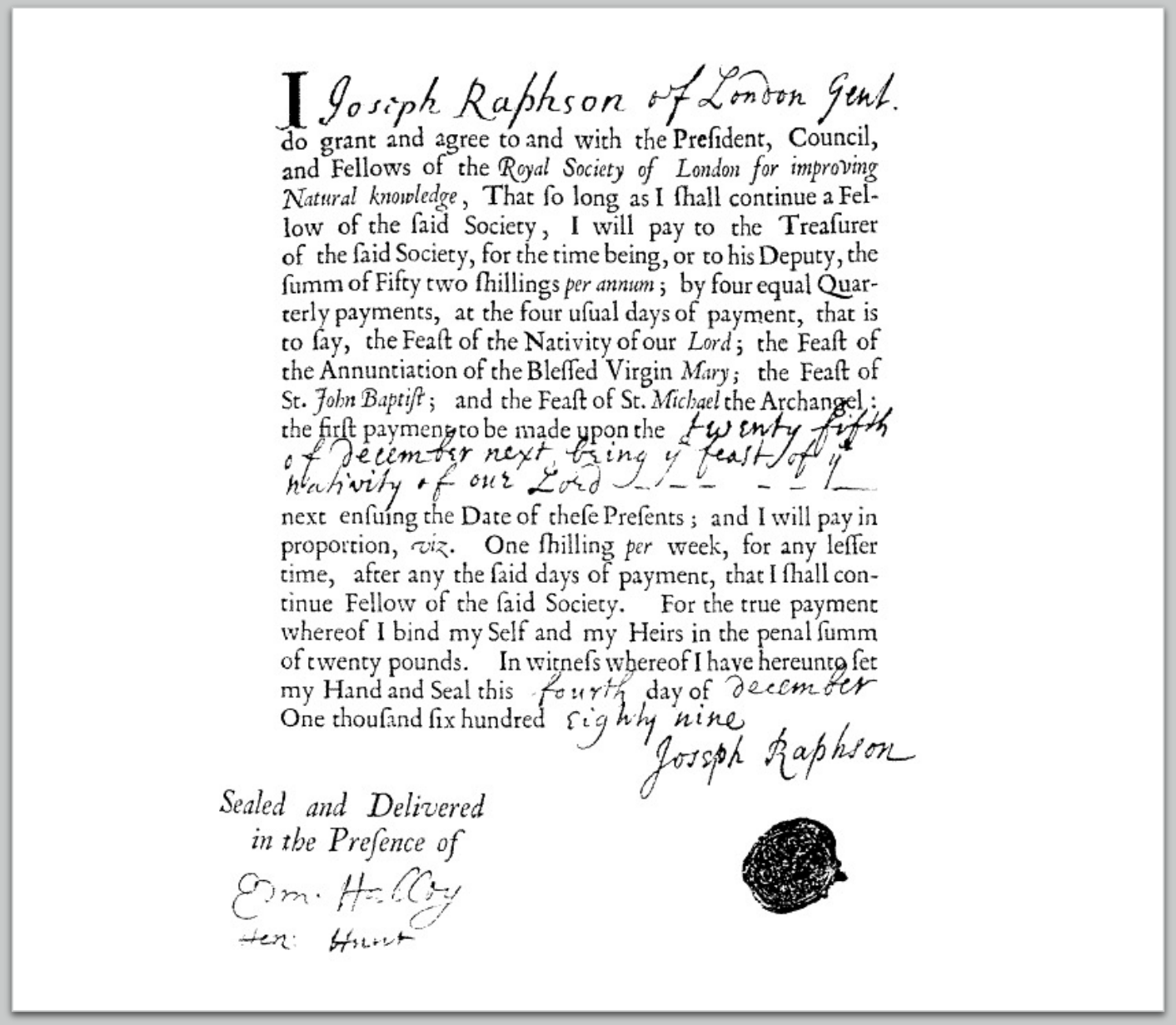
Modern Robotics Ch 6

# Administrivia

- Exam 2 is next week!
  - Topics will be on forward, inverse, and velocity kinematics

# Who is Joseph Raphson?

- Joseph Raphson (1648 – 1715) was an English mathematician known best for the **Newton–Raphson method**
- Raphson's most notable work is *Analysis Aequationum Universalis* (1690), which approximates the roots of an equation
- Isaac Newton had developed a very similar formula in his Method of Fluxions, written in 1671, but published in 1736
- Raphson's method is simpler than Newton's, so Raphson's version is generally used in textbooks today



I Joseph Raphson of London Gent.
do grant and agree to and with the Preſident, Council, and Fellows of the *Royal Society of London for improving Natural knowledge*, That ſo long as I ſhall continue a Fellow of the ſaid Society, I will pay to the Treaſurer of the ſaid Society, for the time being, or to his Deputy, the ſumm of Fifty two ſhillings *per annum*; by four equal Quarterly payments, at the four uſual days of payment, that is to ſay, the Feaſt of the Nativity of our *Lord*; the Feaſt of the Annuntiation of the Bleſſed Virgin *Mary*; the Feaſt of St. *John Baptiſt*; and the Feaſt of St. *Michael* the Archangel; the firſt payment to be made upon the twenty fifth of December next being y feast of y Nativity of our Lord
next enſuing the Date of theſe Preſents; and I will pay in proportion, *viz.* One ſhilling *per* week, for any leſſer time, after any the ſaid days of payment, that I ſhall continue Fellow of the ſaid Society. For the true payment whereof I bind my Self and my Heirs in the penal ſumm of twenty pounds. In witneſs whereof I have hereunto ſet my Hand and Seal this fourth day of December One thouſand ſix hundred eighty nine
Joseph Raphson

*Sealed and Delivered in the Preſence of*
Edm. Halley
Hen: Hunt

Credit: Wikipedia

# Inverse Kinematics

- **Forward Kinematics:** computes the end-effector position from joint angles:

- **Inverse Kinematics:** computes the possible joint angles from the pose of the end-effector

# Numerical Inverse Kinematics

- Given $T(\theta)$, find solutions $\theta$ that satisfy $T(\theta) = X$

- When the **analytic solution** is hard or impossible to come by, we **numerically** solve $T(\theta) - X = 0$

# Newton-Raphson Methods (1)

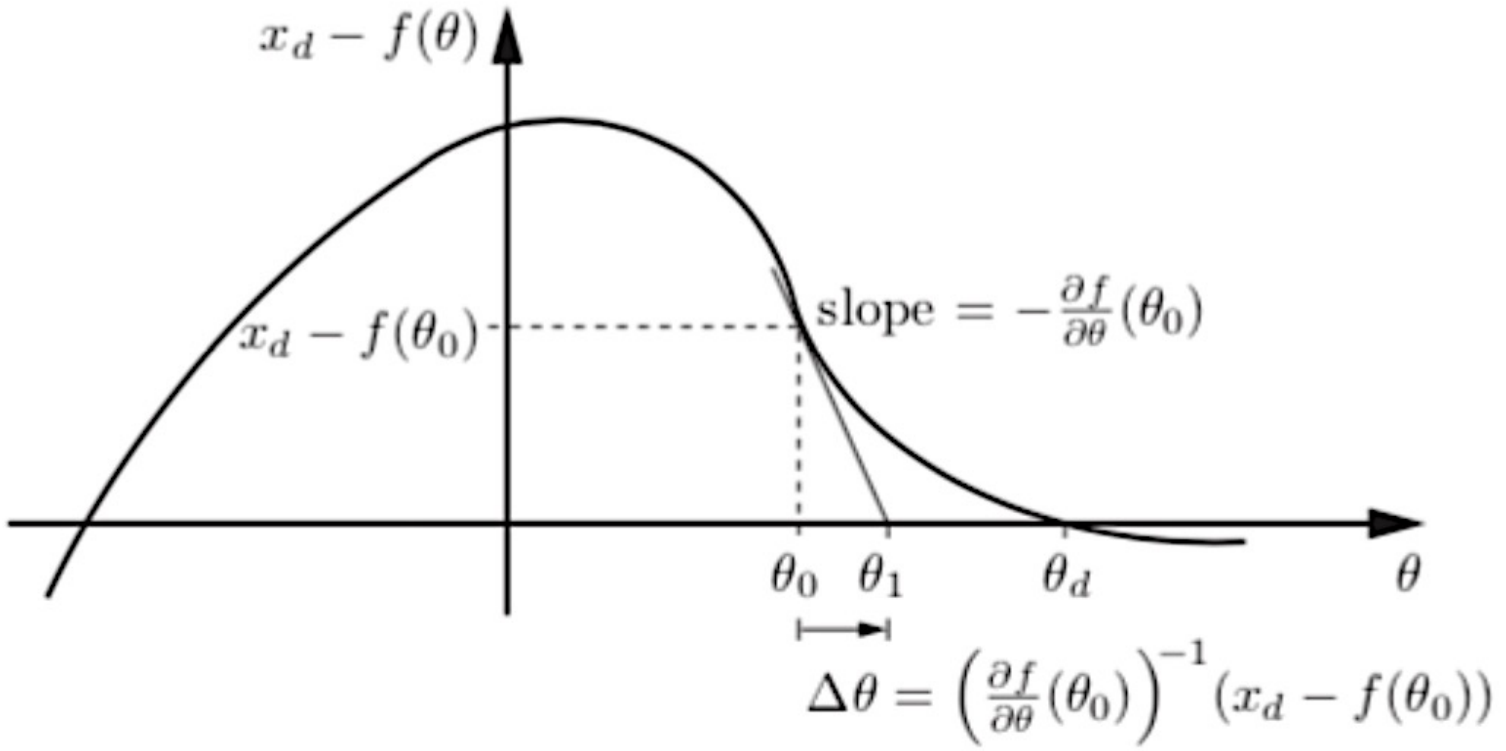
# Newton-Raphson Method (1.5)

# Newton-Raphson Method (2)

# Numerical IK

# Numerical Inverse Kinematics

- If the forward kinematics is linear in $\theta$ (i.e., h.o.t. are zero), then $\theta^1$ exactly satisfies $x_d = f(\theta^1)$
- If the forward kinematics is not linear in $\theta$ (as is usually the case), then $\theta^1$ should be closer to the root than $\theta^0$, but will require more iterations



# Recall: Non-invertible Jacobians and the Pseudo-Inverse

- $J(\theta^0)$ may be singular ($\det\left(J(\theta^0)\right) = 0$) or may not be square
  - Non-invertible! Instead use Pseudo-Inverse!
- The Moore-Penrose pseudo-inverse for $J \in \mathbb{R}^{m \times n}$ is denoted $J^\dagger \in \mathbb{R}^{n \times n}$
- Consider equation: $Jy = z$. We want to find the solution: $y^* = J^\dagger z$, such that:
  - One solution if $n = m$ and J is full rank
    - $y^* = J^{-1}z$
  - Many solutions if $n > m$
    - $y^*$ exactly satisfies $Jy^* = z$, and gives the minimal norm solution ($\|y^*\| \leq \|y\|$)
  - No solutions if $n < m$ and z is not in the span of J
    - If no solution, $y^*$ minimizes the two-norm of the error: $\|Jy^* - z\| \leq \|J\tilde{y} - z\|$ for all $\tilde{y}$

# The Pseudo Inverse

- When J is full column rank (n<m, tall):
- When J is full row rank (n>m, wide):
- When n=m and full rank:

# Using the Newton-Raphson Method for IK

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4\times4}$or a coordinate vector

# Using the Newton-Raphson Method for IK

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4\times4}$or a coordinate vector
- The error vector $e = x_d - f(\theta^i)$, represents the update needed to go from the current guess to the desired end-effector configuration (after being multiplied by the inverse Jacobian)
- Said otherwise, following the direction $e$ for one second, starting from $f(\theta^i)$, should send us (approximately) to $x_d$

# Using the Newton-Raphson Method for IK

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4\times4}$or a coordinate vector
- The error vector $e = x_d - f(\theta^i)$, represents the update needed to go from the current guess to the desired end-effector configuration (after being multiplied by the inverse Jacobian)
- Said otherwise, following the direction $e$ for one second, starting from $f(\theta^i)$, should send us (approximately) to $x_d$
- In our case, we are given $X \in SE(3)$, and instead of computing $X - T(\theta^i)$, we should compute the **twist** $\mathcal{V}_b$ which, if followed for one second, sends us from $T(\theta^i)$ to $X$

# Numerical Inverse Kinematics with a Twist

- We want to find body twist $\mathcal{V}_b$ to move from $T_{sb}(\theta^i)$ to desired $T_{sd} = X$

- The twist that sends us from $T_{sb}(\theta^i)$ to $X$ satisfies:

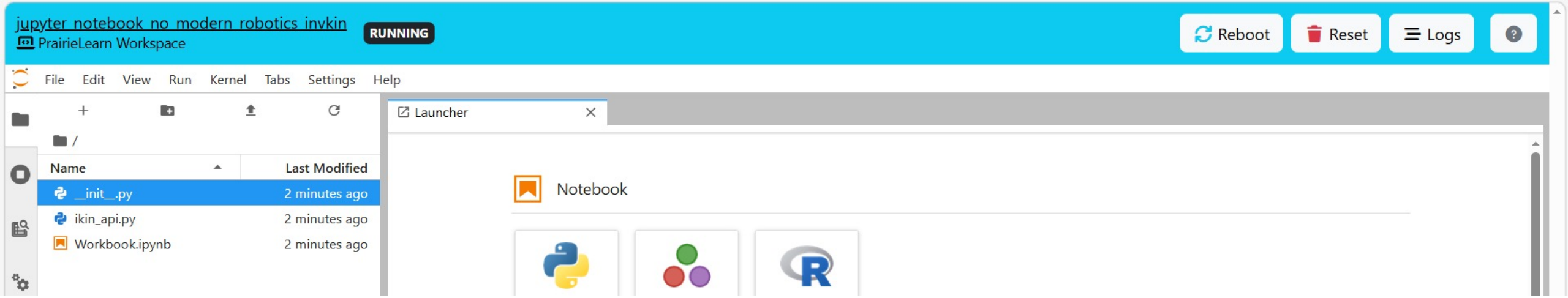
# Numerical Inverse Kinematics: Algorithm

0. Given $X = T_{sd}$ and the forward kinematics map $T_{sb}(\theta)$
   Given tolerances $\epsilon_\omega$ and $\epsilon_v$
1. Initialize $\theta^0$ and set $i = 0$
2. While $\|\omega_b\| > \epsilon_\omega$ or $\|\omega_v\| > \epsilon_v$
   1. Set $[\mathcal{V}_b] = \log T_{sb}^{-1}(\theta^i) T_{sd}$
   2. Set $\theta^{i+1} = \theta^i + J_b^\dagger(\theta^i)\mathcal{V}_b$
   3. Increment $i$

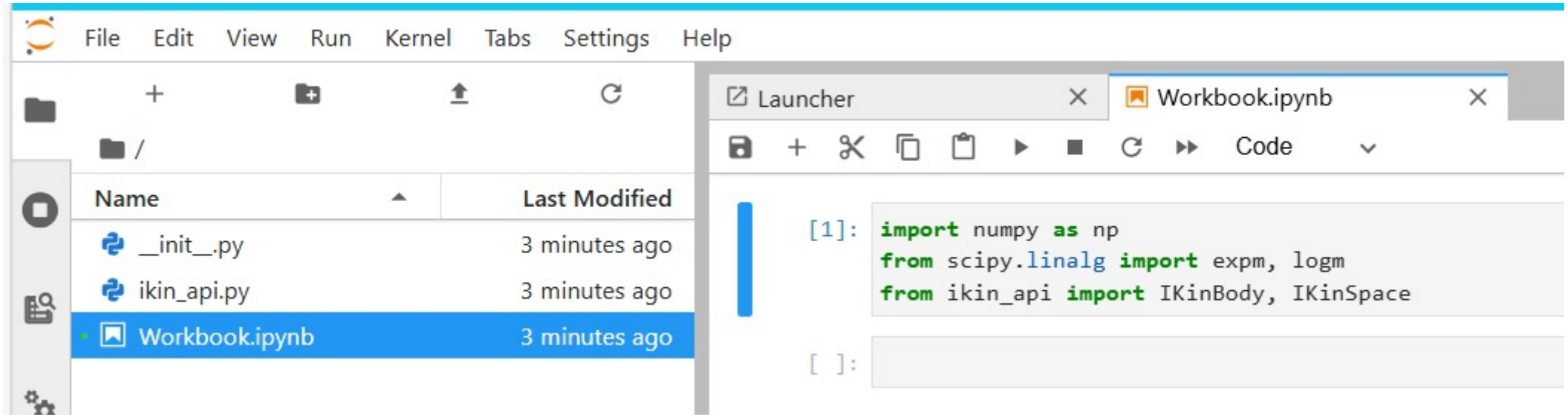
# A few comments

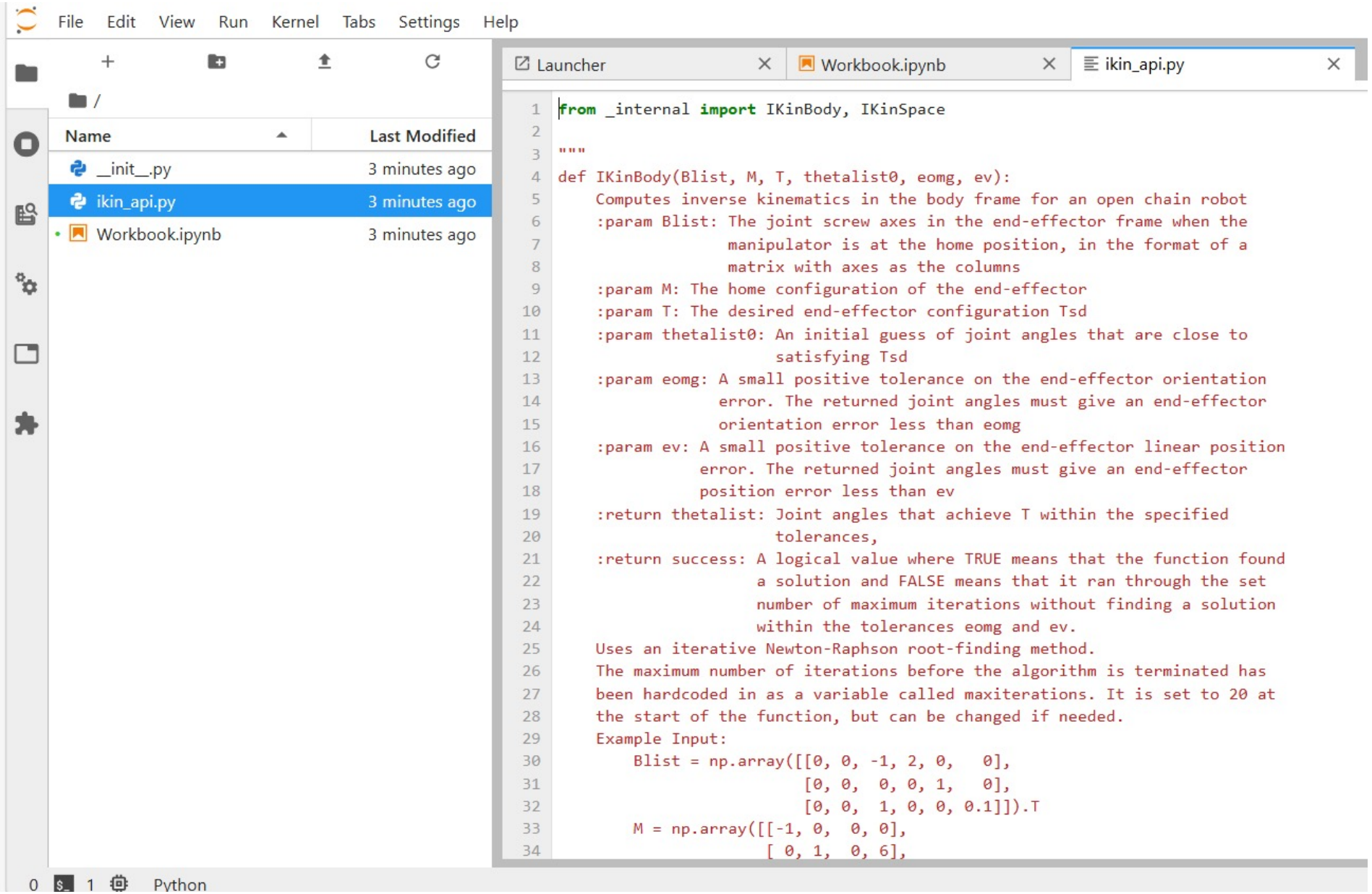
- Re: algorithm on last slide:
  - An equivalent form can be derived in the space frame, using the spatial Jacobian and the spatial twist
- You can also extend this to find sequence of joint angles that will allow the end-effector to follow a trajectory $T_{sb}(t)$
  - Discretize the trajectory $T_{sb}(\theta_k)$ and compute $\theta_k$ such that $T_{sb}(\theta_k) = T_{sb}(t_k)$
  - Initialization is *really* important here!
- For your homework, you will not have to implement Newton-Raphson
  - Must have an understanding of the theory for conceptual questions on the upcoming exam

The Jupyter workspace now provides access to functions for inverse kinematics!
This is what you'll see when you startup the server:




In addition to numpy, expm, and logm, we now import IKinBody and IKinSpace by default:






ikin_api.py explains how to call both functions and has examples

You can ignore __init__.py

[1]:
```
import numpy as np
from scipy.linalg import expm, logm
from ikin_api import IKinBody, IKinSpace
```

[2]:
```
# Desired end effector pose - 4x4 homogeneous transformation matrix
T_1in0 = np.array([[0.64469519, 0.00045421, -0.76443961, -1.72006131], [-0.68757886, 0.43735862, -0.57961430, -3.20838971], [0.33407099, 0

# Initial pose of end effector - 4x4 homogeneous transformation matrix
M = np.array([[1.00000000, 0.00000000, 0.00000000, 2.00000000], [0.00000000, 1.00000000, 0.00000000, -4.00000000], [0.00000000, 0.00000000

# Screw axes - 6xNUM_JOINTS where each column is the screw of each joint (in this example, they are in the fixed frame)
S = np.array([[0.00000000, 0.00000000, 1.00000000, 0.00000000, 1.00000000, 0.00000000], [1.00000000, 0.00000000, 0.00000000, 0.00000000, 0

# Initial guess for what each theta should be - list of length NUM_JOINTS
thetalist0 = np.array([0., 0., 0., 0., 0., 0.])

# Tolerance on error of end effector orientation
eomg = 0.01

# Tolerance on error of end effector position
ev = 0.01
```

[5]:
```
thetalist, success = IKinSpace(S, M, T_1in0, thetalist0, eomg, ev)

# success is True if the error of the solution is less than eomg and ev
# thetalist is of shape (NUM_JOINTS,)
# PrairieLearn expects thetalist is a column vector
thetalist = thetalist.reshape((1,-1)).T
print(thetalist.tolist())
print(thetalist.shape)
```

```
[[-0.35111565922256915], [-0.6162067021942831], [0.34969128094474744], [0.9405509369400207], [0.6555028624350827], [-0.17241756757718962]]
(6, 1)
```

# Summary

- **Inverse Kinematics** gives us a method for finding the joint angles given an end-effector configuration
- This can sometimes be done analytically (geometrically), but this is often difficult so **numerical techniques** are more common in practice
- Introduced the **Newton-Raphson method,** which can be modified to solve inverse kinematics numerically