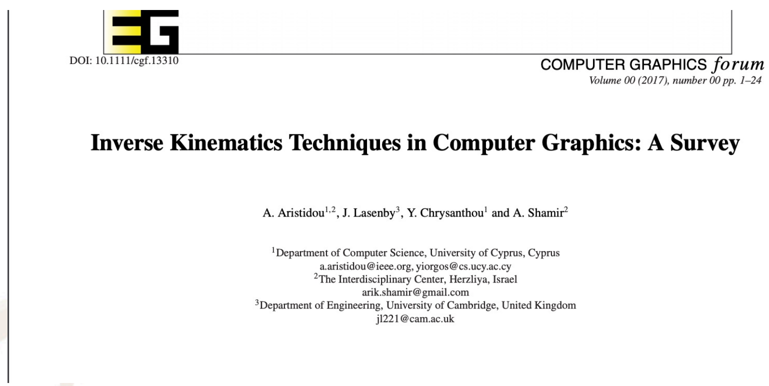
Animation for Computer Games
COMP 477/6311

Prof. Tiberiu Popa

Inverse Kinematics

Acknowledgments

- Material in this lecture based largely on:
- Aristidou, A., Lasenby, J., Chrysanthou, Y., & Shamir, A. (2018, September). Inverse kinematics techniques in computer graphics: A survey. In *Computer Graphics Forum* (Vol. 37, No. 6, pp. 35-58).
- http://www.andreasaristidou.com/publications/papers/IK_survey.pdf



DOI: 10.1111/cgf.13310
COMPUTER GRAPHICS *forum*
Volume 00 (2017), number 00 pp. 1–24

Inverse Kinematics Techniques in Computer Graphics: A Survey

A. Aristidou[1,2], J. Lasenby[3], Y. Chrysanthou[1] and A. Shamir[2]

[1]Department of Computer Science, University of Cyprus, Cyprus
a.aristidou@ieee.org, yiorgos@cs.ucy.ac.cy
[2]The Interdisciplinary Center, Herzliya, Israel
arik.shamir@gmail.com
[3]Department of Engineering, University of Cambridge, United Kingdom
jl221@cam.ac.uk

Jacobian methods

- Buss, S. R. (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, *17*(1-19), 16.

Forward Kinematics

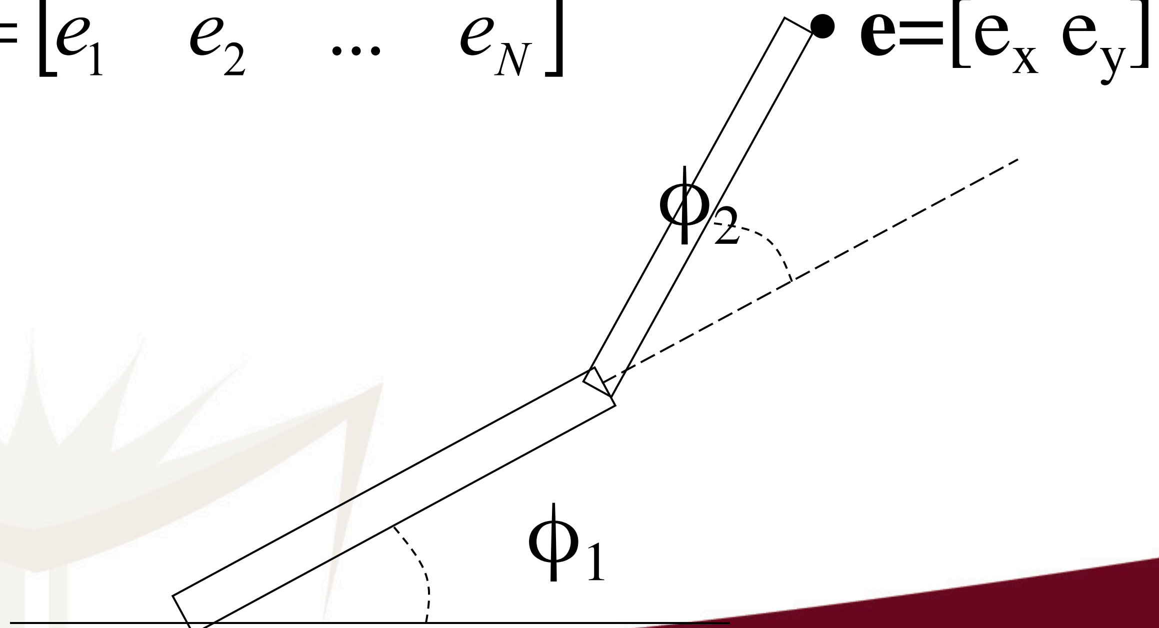
- We will use the vector:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \ldots & \phi_M \end{bmatrix}$$

to represent the array of M joint DOF values

- We will also use the vector:

$$\mathbf{e} = \begin{bmatrix} e_1 & e_2 & \ldots & e_N \end{bmatrix}$$



Forward & Inverse Kinematics

- The forward kinematic function computes the world space end effector DOFs from the joint DOFs:

$$\mathbf{e} = f(\mathbf{\Phi})$$

- The goal of inverse kinematics is to compute the vector of joint DOFs that will cause the end effector to reach some desired goal state

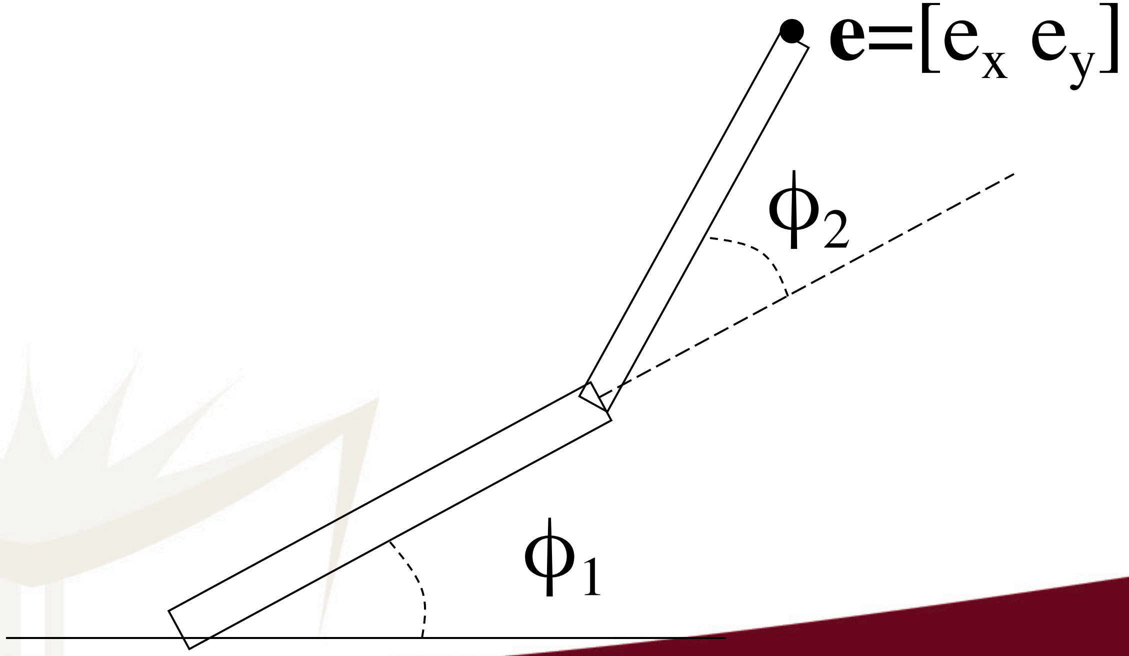
$$\mathbf{\Phi} = f^{-1}(\mathbf{e})$$

Jacobian methods

- Start on the whiteboard…

Jacobians

- Let's say we have a simple 2D robot arm with two 1-DOF rotational joints:



Jacobians

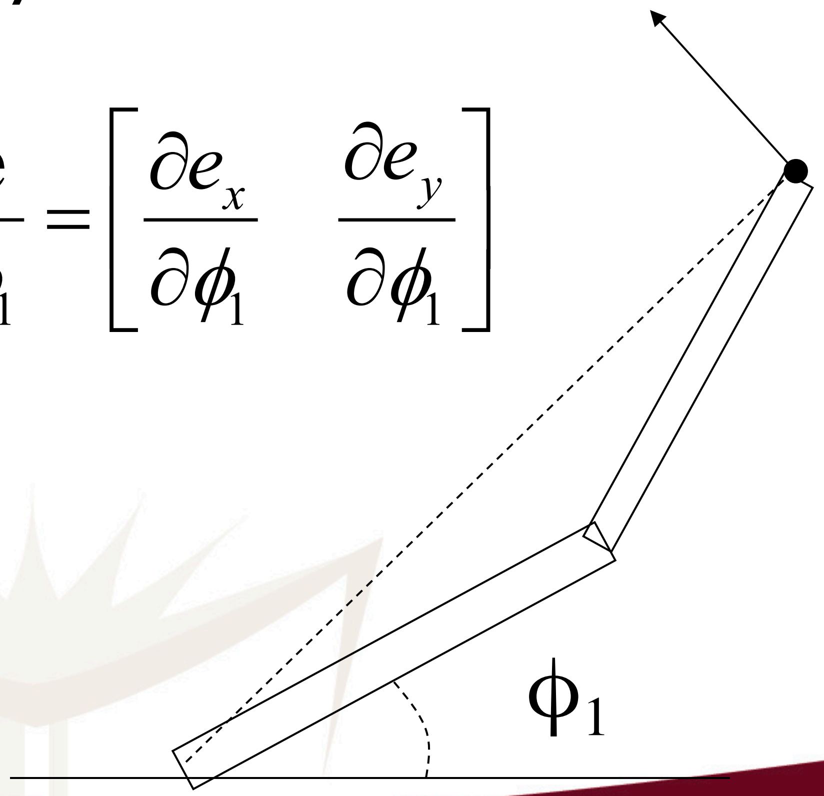
- The Jacobian matrix J(**e**,**Φ**) shows how each component of **e** varies with respect to each joint angle

$$J(\mathbf{e},\mathbf{\Phi}) = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_x}{\partial \phi_2} \\ \frac{\partial e_y}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_2} \end{bmatrix}$$

Jacobians

- Consider what would happen if we increased ϕ_1 by a small amount. What would happen to **e** ?

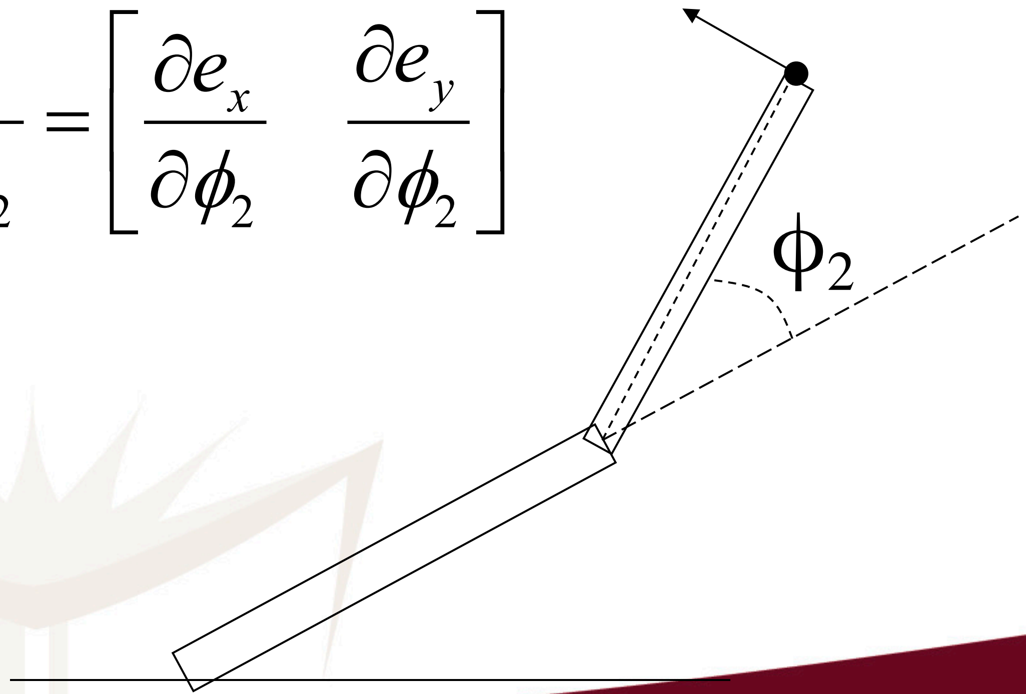
$$\frac{\partial \mathbf{e}}{\partial \phi_1} = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_1} \end{bmatrix}$$



Jacobians

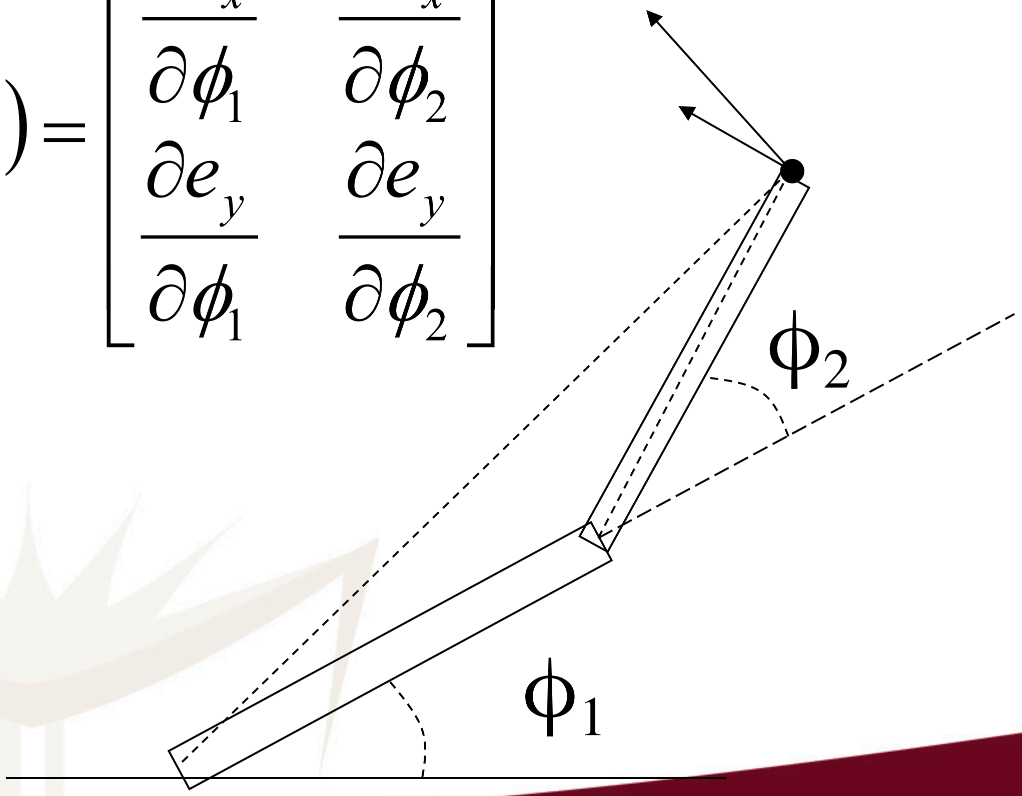
- What if we increased ϕ_2 by a small amount?

$$\frac{\partial \mathbf{e}}{\partial \phi_2} = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_2} & \frac{\partial e_y}{\partial \phi_2} \end{bmatrix}$$



Jacobian for a 2D Robot Arm

$$J(\mathbf{e},\mathbf{\Phi}) = \begin{bmatrix} \dfrac{\partial e_x}{\partial \phi_1} & \dfrac{\partial e_x}{\partial \phi_2} \\ \dfrac{\partial e_y}{\partial \phi_1} & \dfrac{\partial e_y}{\partial \phi_2} \end{bmatrix}$$



Incremental Change in Pose

- Taylor series strikes again

- $\boldsymbol{e}(\boldsymbol{\phi} + \boldsymbol{\Delta\phi}) \approx \boldsymbol{e}(\boldsymbol{\phi}) + \frac{\boldsymbol{de}}{\boldsymbol{d\phi}} \cdot \boldsymbol{\Delta\phi}$

$$\Delta \mathbf{e} \approx \frac{d\mathbf{e}}{d\mathbf{\Phi}} \cdot \Delta\mathbf{\Phi} = J\left(\mathbf{e}, \mathbf{\Phi}\right) \cdot \Delta\mathbf{\Phi} = \mathbf{J} \cdot \Delta\mathbf{\Phi}$$

Incremental Change in Effector

$$\Delta \mathbf{e} \approx \mathbf{J} \cdot \Delta \mathbf{\Phi}$$

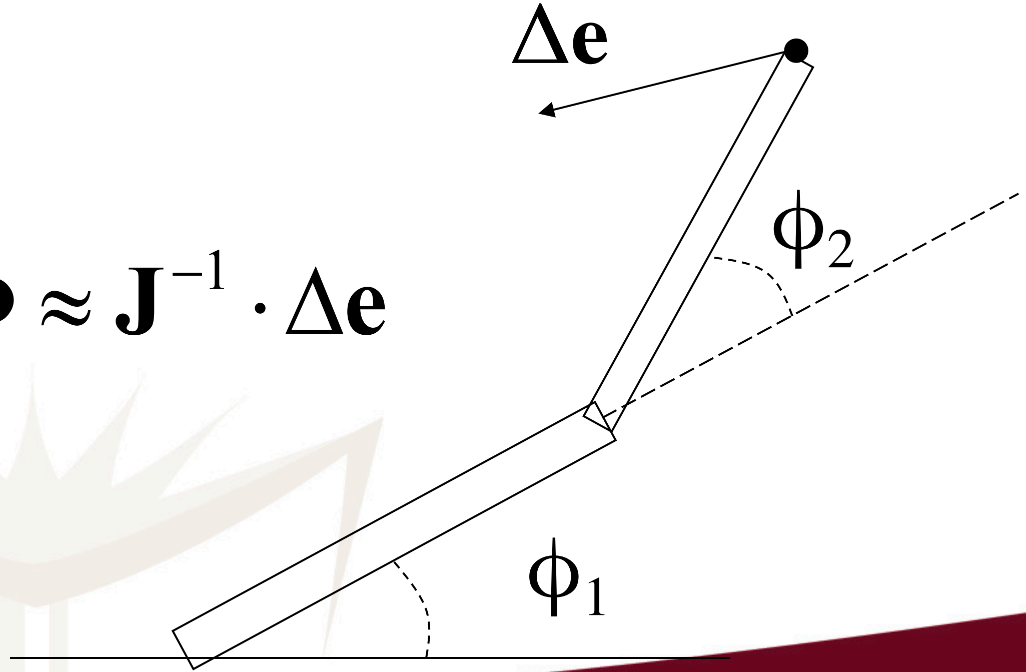
$$so:$$

$$\Delta \mathbf{\Phi} \approx \mathbf{J}^{-1} \cdot \Delta \mathbf{e}$$

Incremental Change in e

- Given some desired incremental change in end effector configuration Δ**e**, we can compute an appropriate incremental change in joint DOFs Δ**Φ**

$$\Delta\Phi \approx J^{-1} \cdot \Delta e$$



Basic Jacobian IK Technique

while ($\mathbf{e}$ is too far from $\mathbf{g}$) {

Compute $J(\mathbf{e},\boldsymbol{\Phi})$ for the current pose $\boldsymbol{\Phi}$

Compute J^{-1} // invert the Jacobian matrix

$\Delta\mathbf{e} = \beta(\mathbf{g} - \mathbf{e})$ // pick approximate step to take

$\Delta\boldsymbol{\Phi} = J^{-1} \cdot \Delta\mathbf{e}$// compute change in joint DOFs

$\boldsymbol{\Phi} = \boldsymbol{\Phi} + \Delta\boldsymbol{\Phi}$ // apply change to DOFs

Compute new $\mathbf{e}$ vector // apply forward

// kinematics to see

// where we ended up

}

What's up with beta

- Remember that forward kinematics is a nonlinear function (as it involves sin's and cos's of the input variables)
- This implies that we can only use the Jacobian as an approximation that is valid near the current configuration
- Therefore, we must repeat the process of computing a Jacobian and then taking a small step towards the goal until we get to where we want to be

Potential problems?

- Jacobian can be non-square or rank deficient

Pseudo-Inverse

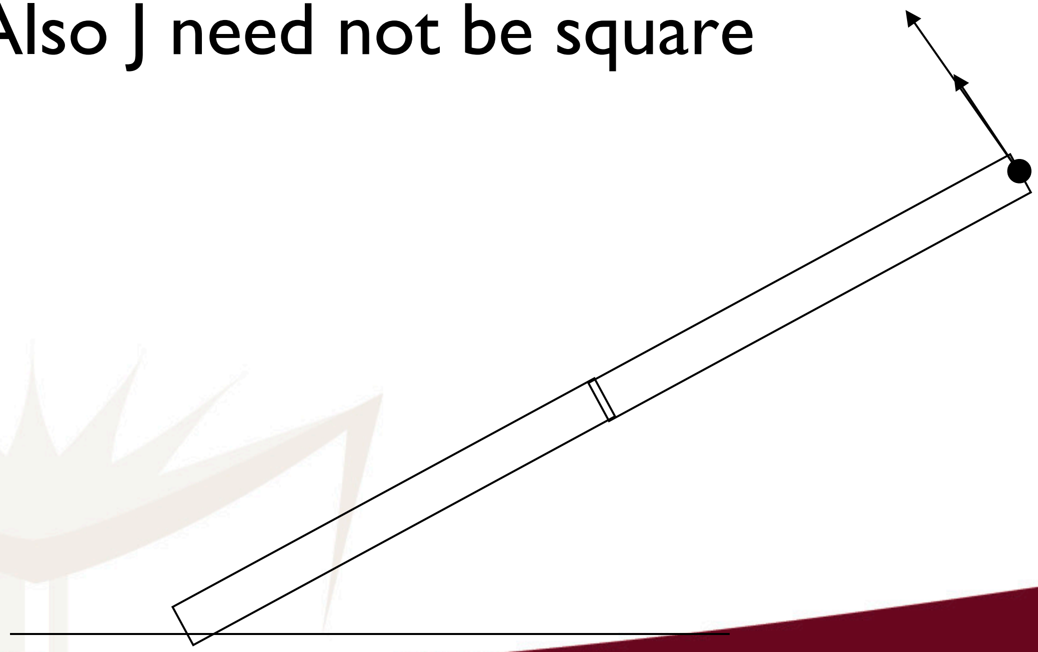
- If we have a non-square we can try using the *pseudoinverse*:

$$\mathbf{J}^{*}=\mathbf{J}^{T}\,(\mathbf{J}^{T}\mathbf{J})^{-1}$$

- This is a method for finding a matrix that effectively inverts a non-square matrix

Degenerate Cases

- Occasionally, we will get into a configuration that suffers from degeneracy
- If the derivative vectors line up, they lose their linear independence
- Also J need not be square



Jacobian Transpose

- With the Jacobian transpose (JT) method, we can just loop through each DOF and compute the change to that DOF directly
- With the inverse (JI) or pseudo-inverse (JP) methods, we must first loop through the DOFs, compute and store the Jacobian, invert (or pseudo-invert) it, then compute the change in DOFs, and then apply the change
- The JT method is far friendlier on memory access & caching, as well as computations
- However, if one prefers quality over performance, the JP method might be better…

Non-degenerate solutions

- Jacobian is not always invertible
- Both transpose and pseudo-inverse are heuristics → can have failure cases
- Solutions:
  - Second order Taylor series (Newton's method)
  - Damped Least Squares

Damped Least Square

- Very useful optimization technique in general

Université Concordia University

Incremental Change in Effector

$$\Delta \mathbf{e} \approx \mathbf{J} \cdot \Delta \mathbf{\Phi}$$

$$so:$$

$$\Delta \mathbf{\Phi} \approx \mathbf{J}^{-1} \cdot \Delta \mathbf{e}$$

Incremental Change in Effector

$$\underset{\Delta\phi}{\operatorname{argmin}} ||\Delta e - J \cdot \Delta\phi||_2^2$$

If J is not full ranked, have a null space → infinitely many solutions
Not necessarily at 0

Incremental Change in Effector

$$\underset{\Delta\phi}{\operatorname{argmin}} ||\Delta e - J \cdot \Delta\phi||_2^2$$

If J is not full ranked, have a null space → infinitely many solutions
Not necessarily at 0

Levenberg Marquardt algorithm
Very useful
Main idea: chose a minimizer that is as small as possible

$$\underset{\Delta\phi}{\operatorname{argmin}} ||\Delta e - J \cdot \Delta\phi||_2^2 + \lambda^2 ||\Delta\phi||_2^2$$

Lambda is the damping constant
Why we need Lambda?

Incremental Change in Effector

$$\underset{\Delta\phi}{\operatorname{argmin}} \|\Delta e - J \cdot \Delta\phi\|_2^2$$

$$\boldsymbol{\Delta\theta} = (J^T J)^{-1} J^T \vec{\mathbf{e}}.$$

$$\underset{\Delta\phi}{\operatorname{argmin}} \|\Delta e - J \cdot \Delta\phi\|_2^2 + \lambda^2 \|\Delta\phi\|_2^2$$

$$\boldsymbol{\Delta\theta} = (J^T J + \lambda^2 I)^{-1} J^T \vec{\mathbf{e}}.$$