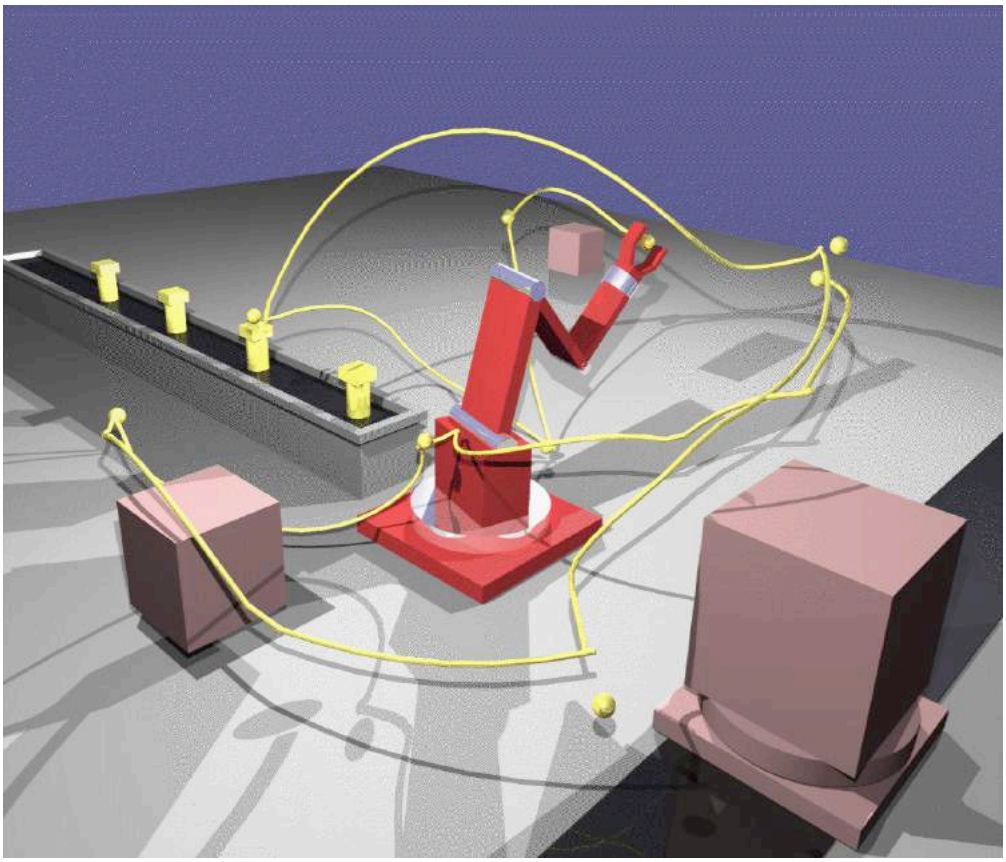
# **Goal-Oriented Graphics Animation**:

## An articulated object

## Contents

# 1 Overview

In this assignment, you will write a program that creates an animation of the motion of a robot arm. The arm will move from an initial position to a set of multiple goal positions. As the arm moves, it should avoid collisions with the various obstacles in the scene. The animation method to be implemented is described in Example 2 of Breen's paper [1]. Figure 1 shows an example output for this animation.



(a)

Figure 1: Goal-oriented animation: The motion path for a particle.

The assignment is divided into two parts. In the first part, we will implement the forward-kinematics function of the robot arm, draw the scene in 3-D, and use the inverse-kinematics function to draw the robot and try out a few poses given a set of joint angles of your choice. In the second part of the assignment, we will implement the animation of the arm using inverse kinematics and cost-function minimization.

These notes describe the main concepts and mathematical background needed to complete

the assignment. Further reading might be needed to fill in some gaps. These notes were heavily based on Steve Rotenberg's lectures slides from the CSE 169 Computer Animation course[1] and from the online tutorial by Alan Zucconi [2].

# 2 Part I: Forward kinematics and drawing the arm

## 2.1 Forward kinematics of the robot arm

We want to define the function $\mathbf{e}$ that controls the arm's pose and the end-effector position (and pose), i.e., the forward-kinematics function. This function takes the vector of joint angles $\mathbf{\Phi}$ as input and returns the location of the end-effector, $\mathbf{e}(\mathbf{\Phi})$. Here, $\mathbf{\Phi} = (\phi_1, \phi_2, \ldots, \phi_M)^\mathsf{T}$ represent the $M$ joint angles of the kinematic object and $\mathbf{e} = (e_1, e_2, \ldots, e_N)^\mathsf{T}$ represent the $N$ degrees-of-freedom (DOF) of the end effector. The forward-kinematics function $f$ computes the position of the end effector given the joint-angle configuration, i.e.:

$$\mathbf{e} = f(\mathbf{\Phi}). \tag{1}$$

In many applications, $\mathbf{e}$ has 6 DOFs, three for 3-D rotation, and three for its 3-D location. In this assignment, we will assume that we are interested only in the end-effector's position. As a result, $N = 3$ and $\mathbf{e} = (e_x, e_y, e_z)^\mathsf{T}$. Also, the robot arm in Figure 1 has 4 joint angles, i.e., $\mathbf{\Phi} = (\phi_1, \phi_2, \phi_3, \phi_4)^\mathsf{T}$.

The robot arm's parts and joints are illustrated in the diagram in Figure 2, from which we can derive the arm's forward-kinematic function $\mathbf{e}(\mathbf{\Phi})$. The base of the arm is a *cylindrical* joint (joint 1) that rotates about the $z_0$-axis. All other joints are *elbow* joints that rotate about their y-axis. We assume that each joint is centered at a local coordinate frame whose z-axis is aligned with the corresponding arm part that attaches to the joint. Figure 2(b) shows the 4 coordinate frames of the robot arm, and $x_0y_0z_0$ is the *world* coordinate frame. The joint center $\mathbf{p}_5$ is the end-effector position $\mathbf{e}$.

To build the forward-kinematics equation, our first step is to build the individual change-of-coordinate transformations (i.e., rigid-body transformation) that convert points from frame $i$ to frame $i + 1$. The individual transformations are described next.

[1] `https://cseweb.ucsd.edu/classes/sp16/cse169-a/sessions.html`

[2] `https://www.alanzucconi.com/2017/04/10/robotic-arms/`

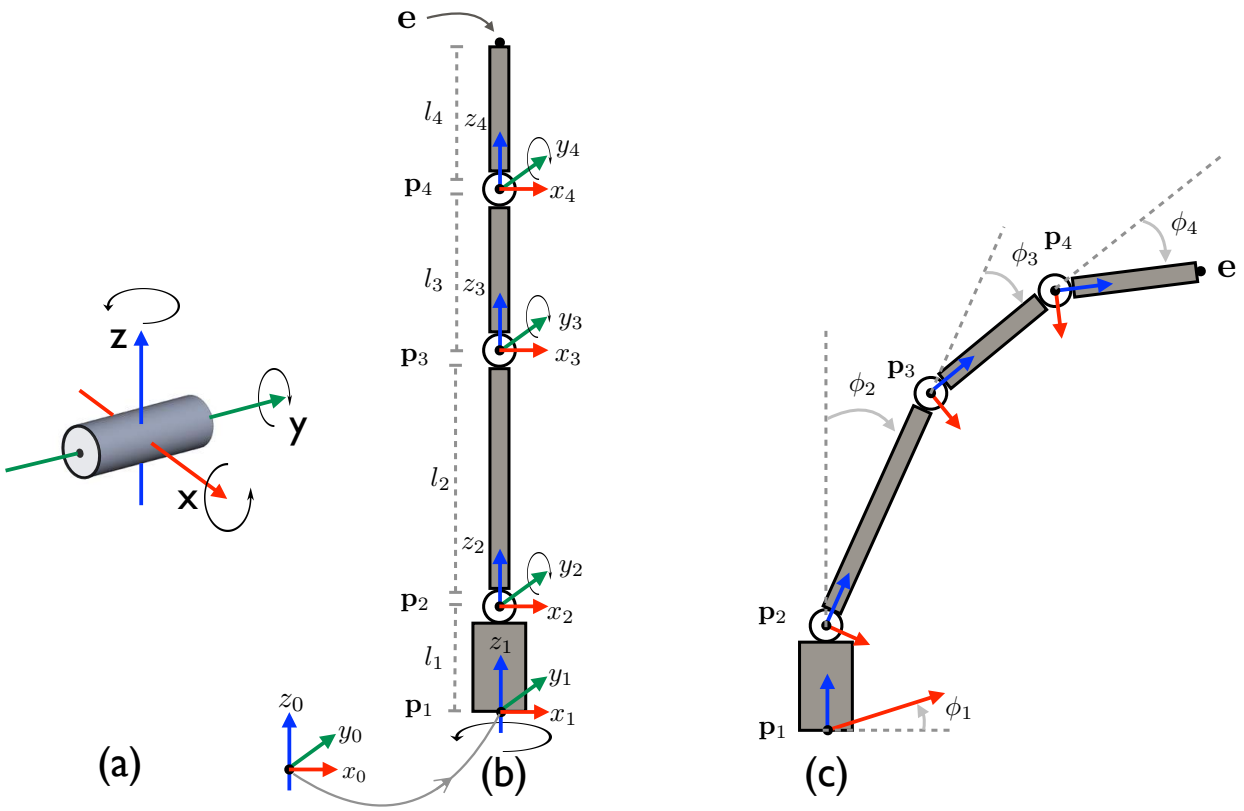



Figure 2: (a) Elbow join with y-axis rotation. The arm part that connects to the joint is aligned with that part's z-axis. (b) Robot arm in its home (i.e., zero) configuration. (c) Part $l_1$ rotates about the $z_1$-axis. All other parts have elbow joints and rotate about the part's $y$-axis. In this diagram the joint center $\mathbf{p}_5$ is the end-effector position $\mathbf{e}$.

### 2.1.1 Individual coordinate-frame transformations

The first transformation converts coordinates from frame 0 (world frame) to frame 1. Frame 1 is given by a rotation about the $z_0$-axis followed by a translation by the location of the joint $\mathbf{p}_1$ in world coordinates, which is given by:

$$T_{0,1} = \begin{bmatrix} R_z(\phi_0) & \mathbf{p}_1 \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} \cos\phi_0 & -\sin\phi_0 & 0 & 0 \\ \sin\phi_0 & \cos\phi_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2)$$

In the case of Equation 2, $\mathbf{p}_1 = (0,0,0)^\mathsf{T}$. This is the location of the joint center corresponding to the base of the arm. If we want the base of the arm to be elsewhere, we change the coordinates of $\mathbf{p}_1$ in the transformation matrix.

The transformations the next three subsequent coordinate frames of the arm are rotations about the part's $y_i$-axes, followed by the translation of size $l_i$ along that part's $z_i$-axis, i.e.:

$$T_{1,2} = \begin{bmatrix} R_y(\phi_1) & \mathbf{t}_z(l_1) \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} \cos\phi_1 & 0 & \sin\phi_1 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi_1 & 0 & \cos\phi_1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

$$T_{2,3} = \begin{bmatrix} R_y(\phi_2) & \mathbf{t}_z(l_2) \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} \cos\phi_2 & 0 & \sin\phi_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi_2 & 0 & \cos\phi_2 & l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4}$$

and

$$T_{3,4} = \begin{bmatrix} R_y(\phi_3) & \mathbf{t}_z(l_3) \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} \cos\phi_3 & 0 & \sin\phi_3 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi_3 & 0 & \cos\phi_3 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

Finally, the last transformation of the kinematic chain, $T_{45}$, takes us from frame 4 to the end-effector frame (i.e., frame 5). Because we decided that our end effector has no orientation in space, the rotation submatrix is just the identity matrix. It also means also the end-effector frame is aligned with frame 4. The final transformation is:

$$T_{4,5} = \begin{bmatrix} \mathbb{1}_{3x3} & \mathbf{t}_z(l_4) \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

Note that $T_{45}$ takes us from frame 4 to the frame of the end effector (i.e., frame 5). Because we decided that our end effector has no orientation in space, the rotation submatrix is just the identity matrix. This means that the end-effector frame is aligned with frame 4.

### 2.1.2 Frames in world coordinates

The final mathematical step for creating the forward-kinematics equation is to represent each joint-frame in world coordinates. This representation is based on the fact that we can combine

consecutive transformation to compose new transformations. For example, the coordinate frame 2, expressed in world coordinates is given by $T_{0,2} = T_{0,1}T_{1,2}$. The composition of transformations is illustrated in Figure 3.

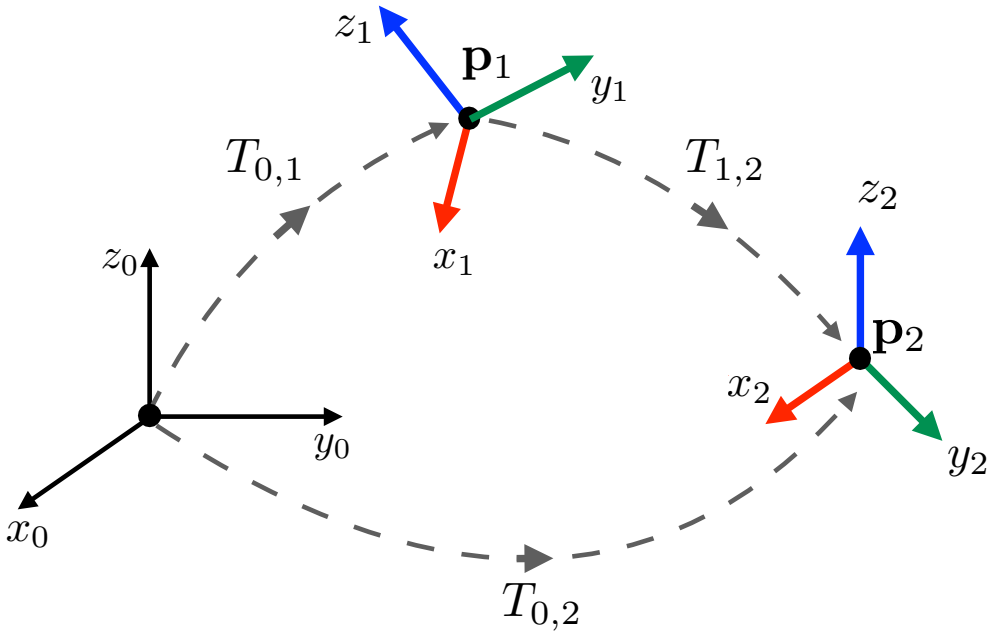



Figure 3: Coordinate frame 2 in world coordinates (frame 0) is given by a composed transformation formed by multiplying the matrices of the transformation chain, i.e., $T_{0,2} = T_{0,1}T_{1,2}$.

The coordinate frames at the joint points with respected to world coordinates are given by:

$$
\begin{aligned}
T_{0,1} &= \begin{bmatrix} R_{0,1} & \mathbf{p}_1 \\ \mathbf{0} & 1 \end{bmatrix}, \\
T_{0,2} = T_{0,1}T_{1,2} &= \begin{bmatrix} R_{0,2} & \mathbf{p}_2 \\ \mathbf{0} & 1 \end{bmatrix}, \\
T_{0,3} = T_{0,2}T_{2,3} = T_{0,1}T_{1,2}T_{2,3} &= \begin{bmatrix} R_{0,3} & \mathbf{p}_3 \\ \mathbf{0} & 1 \end{bmatrix}, \\
T_{0,4} = T_{0,3}T_{3,4} = T_{0,1}T_{1,2}T_{2,3}T_{3,4} &= \begin{bmatrix} R_{0,4} & \mathbf{p}_4 \\ \mathbf{0} & 1 \end{bmatrix}, \\
T_{0,5} = T_{0,4}T_{4,5} = T_{0,1}T_{1,2}T_{2,3}T_{3,4}T_{4,5} &= \begin{bmatrix} R_{0,5} & \mathbf{p}_5 \\ \mathbf{0} & 1 \end{bmatrix}.
\end{aligned} \tag{7}
$$

Finally, the joint center $\mathbf{p}_i$ of the robot arm is the translation component of the transformation $T_{0i}$ (i.e., the elements in the first 3 rows of the the last column).

### 2.1.3 Forward-kinematics function

The implementation of the forward-kinematics function is listed in Algorithm 1.

**Algorithm 1** Forward Kinematics Function

**function** e($\mathbf{\Phi}$)
  $l_1, \ldots, l_4 \leftarrow$ Lengths of each part of the arm
  $T_{i-1,i}$, for $i = 1, \ldots, 4$. ▷ Construct individual transformations (Eqs. 2 to 6)
  **for** $i = 2, \ldots, 5$ **do**
    $T_{0,i} \leftarrow T_{0,i-1} * T_{i-1,i}$ ▷ Compute frames in world coordinates (Eq. 7)
  **end for**
  **for** $i = 1, \ldots, 5$ **do**
    $\mathbf{p}_i \leftarrow$ lastColumnOf($T_{0,i}$) ▷ Obtain joint center from transformation matrix
  **end for**
  **return** $(T_{0,1}, \ldots, T_{0,5})$ ▷ Return all frames w.r.t. the world frame
**end function**

## 2.2 Drawing the robot arm

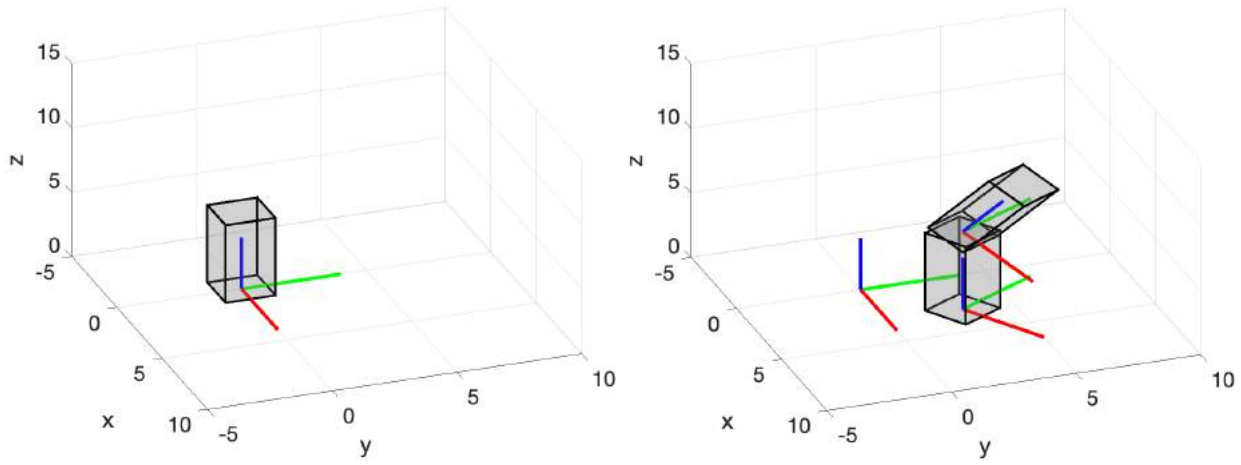
To draw the robot arm, we must draw each part at their associated joint center, $\mathbf{p}_i$. This location is the 3-D translation of the coordinate-frame transformation. Also, the part must be rotated according to the part's coordinate-frame transformation. To place the part at the correct location and orientation, we transform each part using the transformations calculated in the forward-kinematics equations (Equation 7).

### 2.2.1 Example of drawing parts using coordinate-frame transformations

In this example, we draw two parts of an articulated object. Here, each part is represented by a rectangular prism of the same size. First, we need to construct the part, which will have the center of its base located at the origin of the world coordinate frame. Also, we want the frame's z-axis to be aligned with the part's longest side. The following set of vertices

describe the rectangular prism shown in Figure 4(a).

$$v = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 10 & 10 & 10 & 10 & 0 & 0 \end{bmatrix}. \tag{8}$$




(a) (b)

Figure 4: (a) A rectangular prism. The center of its base is located at the origin. The longest size of the prism is aligned with the z-axis of the world coordinate frame. (b) A two-part arm built with rectangular prisms.

We will assume that the object's base joint is located at $\mathbf{p}_1 = (3, 3, 0)^\mathsf{T}$ and object's part 1 rotates about the z-axis of frame 1. Let $T_{0,1}$ and $T_{0,2}$ be the coordinate transformations for frame 1 and frame 2. To draw each part in their correct place and orientation, we need only to transform the vertices of the prism by coordinate-frame matrices before we draw the part, e.g., $\texttt{draw3D}\left(T_{0,1} * v\right), \texttt{draw3D}\left(T_{0,2} * v\right)$. Figure 4(b) shows the result of plotting the prism using transformations $T_{0,1}$ and $T_{0,2}$.

# 3 Part II: Animation using inverse kinematics and cost minimization

## 3.1 Cost function

The animation is achieved by using the gradient-descent algorithm to solve the following minimization problem:

$$\hat{\mathbf{\Phi}} = \arg\min_{\Phi} C(\mathbf{\Phi}), \tag{9}$$

where the cost function $C$ is defined by:

$$C(\mathbf{\Phi}) = \underbrace{\|\mathbf{e}(\mathbf{\Phi}) - \mathbf{g}\|}_{\text{goal attraction}} + \underbrace{\sum_{i=1}^{n} \mathcal{F}_R(\|\mathbf{e}(\mathbf{\Phi}) - \mathbf{o}_i\|)}_{\text{obstacle-avoidance penalty}} + \underbrace{\sum_{j=1}^{3} \mathcal{L}(\phi_j)}_{\text{Joint-range limit}} . \tag{10}$$

Here, $\mathbf{g}$ is the *goal location*, $\mathbf{o}_i$ is the location of obstacle $i$. Function $\mathbf{e}(\mathbf{\Phi})$ computes the arm's *forward kinematics* and returns the location of the arm's tip $\mathbf{e} = (e_x, e_y, e_z)^\mathsf{T}$, i.e., the *end-effector*, given the arm's joint angles, $\mathbf{\Phi} = (\phi_1, \phi_2, \phi_3, \phi_4)^\mathsf{T}$. Function $\mathcal{F}_R$ is a collision-avoidance penalty field. It penalizes poses that take the end effector too close to an obstacle, i.e., beyond a pre-defined distance $R$. The third component of Equation 10 limits the range of each joint angle. Function $\mathcal{L}$ is another penalty function. Its value increases as the joint angle $\phi_j$ approaches its maximum or minimum limit. Outside these limits, $\mathcal{L}$ vanishes.

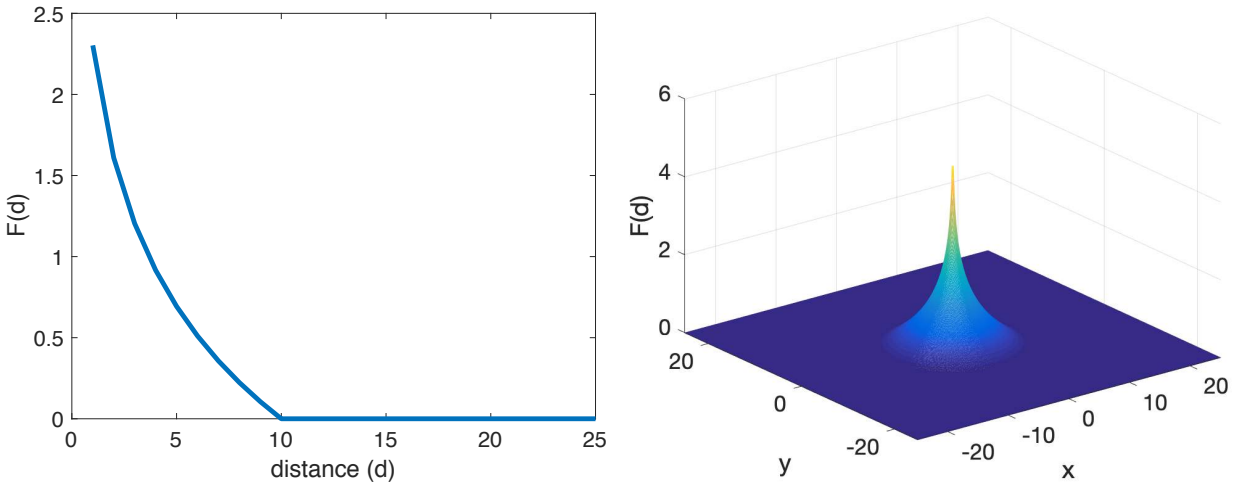
Next, we describe the components of Equation 10 in more detail.

### 3.1.1 Field potential function for obstacle avoidance

The field potential is a *penalty term* that increases its value as the particle approaches an obstacle. It can be defined as follows [1]:

$$\mathcal{F}_R(d) = \begin{cases} \ln(R/d), & 0 < d \leq R, \\ 0, & d > R. \end{cases} \tag{11}$$

An example of the field potential in (11) is shown in Figure 5.




(a) (b)

Figure 5: Penalty field function for $R = 10$. (a) One-dimensional plot of the penalty function. Penalty value increases for distances close to the obstacle. (b) 2-D representation of the field function for $d = \sqrt{x^2 + y^2}$, i.e., distance from any point $(x, y)$ to the origin. When the distance from obstacle is larger than R, the field potential vanishes.

## 3.2 Limit function

The limit function constrains the range of motion of the joints (i.e., angles), and is given by:

$$\mathcal{L}(\phi) = \begin{cases} \ln\left(\delta / \left(\phi - \phi_{\text{min}}\right)\right), & \phi_{\text{min}} < \phi \leq \phi_{\text{min}} + \delta \\ 0, & \phi_{\text{min}} + \delta < \phi < \phi_{\text{max}} - \delta \\ \ln\left(\delta / \left(\phi_{\text{max}} - \phi\right)\right), & \phi_{\text{max}} - \delta \leq \phi < \phi_{\text{max}}, \end{cases} \tag{12}$$

where $\phi$ is the joint angle, $\phi_{\text{min}}$ and $\phi_{\text{max}}$ are the limits of that joint, and $\delta$ is the angular distance from each of the limits after which the limit function vanishes. Figure 6 shows an example of the limit function for $\delta = 45°$, $\phi_{\text{min}} = 90°$, and $\phi_{\text{max}} = 270°$.

## 3.3 Inverse kinematics

In Section 2.1, we learned how we create the forward-kinematics function that moves the robot arm given a set of (known) joint angles. In computer graphics, forward kinematics also helps us to draw the robot arm. However, when what we want is to move the end effector to a specific location, we need to compute the inverse kinematics. To compute the joint-angle

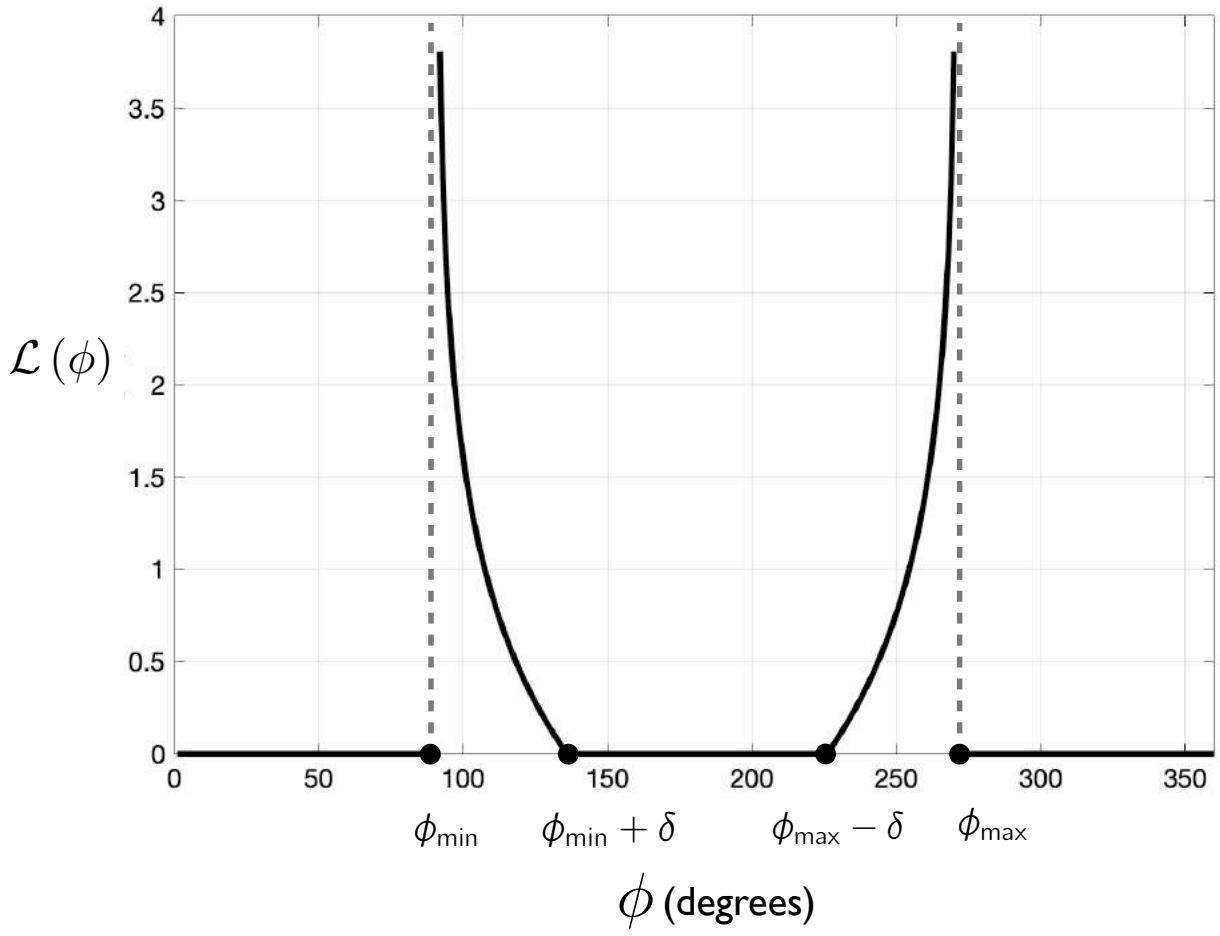



Figure 6: Limit function. One-dimensional plot of the limit function $\mathcal{L}$. For a given joint angle $\phi \in (\phi_{\text{min}}, \phi_{\text{max}})$, the penalty value increases as the joint angle approaches the limits.

configuration for a given goal position of the end effector, we use inverse kinematics, which is the inverse of the forward-kinematics function in Equation 1, i.e.:

$$\mathbf{\Phi} = f^{-1}(\mathbf{e}). \tag{13}$$

However, this function does not usually have a closed-form solution and non-linearity will cause it to have multiple solutions. Instead of trying to solve inverse kinematics directly, we can make progress by using an iterative numerical-minimization method.

## 3.4 Numerical solution to inverse kinematics

One numerical method used for computing inverse kinematics works is based on the gradient-descent approach, and attempts to find the best incremental changes in joint-angle configura-

tion that take the end effector closer a goal position (and orientation). In the gradient-descent method, the best direction for incremental changes in the function variables is the inverse of the gradient. Because the forward-kinematics function is a vector function of vector variables, the "gradient" vector is replaced by the Jacobian matrix.

We want to determine how incremental changes of joint-angle configuration relate to incremental changes of end-effector location, i.e., we want to find out how to relate changes in any component of $\mathbf{\Phi}$ with changes in any component of $\mathbf{e}$. With this relationship at hand, we can derive an iterative algorithm analogous to the gradient-descent to calculate the arm's pose needed to reach a given location.

### 3.4.1 Relating variations of pose to variations of end-effector location

To relate small changes between values of a function and small variations of its variables, we can use derivatives. Indeed, the derivative of forward-kinematics function, $d\mathbf{e}/d\mathbf{\Phi}$, contains all the information we need to know about how to relate changes in any component of $\mathbf{\Phi}$ to changes in any component of $\mathbf{e}$. Because the forward-kinematics function $\mathbf{e}$ is a vector-valued function of the vector-valued variable $\mathbf{\Phi}$, we know from Section B.7 that the derivative of $\mathbf{e}$ is given by the Jacobian. The Jacobian for the robot arm in this assignment is:

$$\frac{\mathrm{d}\mathbf{e}}{\mathrm{d}\mathbf{\Phi}} = J(\mathbf{e}, \mathbf{\Phi}) = \begin{bmatrix} \frac{\partial \mathbf{e}}{\partial \phi_1} & \frac{\partial \mathbf{e}}{\partial \phi_2} & \frac{\partial \mathbf{e}}{\partial \phi_3} & \frac{\partial \mathbf{e}}{\partial \phi_4} \end{bmatrix} = \begin{bmatrix} \frac{\partial e_x}{\partial \phi_1} & \frac{\partial e_x}{\partial \phi_2} & \frac{\partial e_x}{\partial \phi_3} & \frac{\partial e_x}{\partial \phi_4} \\ \frac{\partial e_y}{\partial \phi_1} & \frac{\partial e_y}{\partial \phi_2} & \frac{\partial e_y}{\partial \phi_3} & \frac{\partial e_y}{\partial \phi_4} \\ \frac{\partial e_z}{\partial \phi_1} & \frac{\partial e_z}{\partial \phi_2} & \frac{\partial e_z}{\partial \phi_3} & \frac{\partial e_z}{\partial \phi_4} \end{bmatrix}. \qquad (14)$$

This Jacobian's first column vector, $\partial\mathbf{e}/\partial\phi_1$, describes how the end-effector position changes for small changes in join angle $\phi_1$. Similarly, the second column vector, $\partial\mathbf{e}/\partial\phi_2$, describes how the end-effector position changes for small changes in join angle $\phi_2$. Note that each component of the Jacobian matrix in Equation 14 is equivalent to a 1st-order derivative of a scalar function of a single variable, and can be approximated numerically using finite

differences. The approximated Jacobian is then:

$$J(\mathbf{e}, \mathbf{\Phi}) \approx \begin{bmatrix} \frac{\Delta e_x}{\Delta \phi_1} & \frac{\Delta e_x}{\Delta \phi_2} & \frac{\Delta e_x}{\Delta \phi_3} & \frac{\Delta e_x}{\Delta \phi_4} \\ \frac{\Delta e_y}{\Delta \phi_1} & \frac{\Delta e_y}{\Delta \phi_2} & \frac{\Delta e_y}{\Delta \phi_3} & \frac{\Delta e_y}{\Delta \phi_4} \\ \frac{\Delta e_z}{\Delta \phi_1} & \frac{\Delta e_z}{\Delta \phi_2} & \frac{\Delta e_z}{\Delta \phi_3} & \frac{\Delta e_z}{\Delta \phi_4} \end{bmatrix}. \tag{15}$$

for a small $\Delta\phi_i, i = 1, \ldots, 4$, where each element of the matrix is given by Equation 29.

We now have all the ingredients we need to derive a numerical algorithm similar to the gradient descent to compute inverse kinematics on our robot arm. The algorithm's derivation is analogous to the gradient-descent for scalar functions in Section B.4.

### 3.4.2 How pose change as a function of end-effector position

Let vector $\Delta\mathbf{\Phi}$ be a small change in joint-angle values, which causes the following change in the end effector $\mathbf{e}$:

$$\begin{aligned} \frac{\Delta \mathbf{e}}{\Delta \mathbf{\Phi}} &\approx \frac{\mathrm{d}\mathbf{e}}{\mathrm{d}\mathbf{\Phi}} \\ \Delta \mathbf{e} &\approx \frac{\mathrm{d}\mathbf{e}}{\mathrm{d}\mathbf{\Phi}} \cdot \Delta \mathbf{\Phi} \\ &\approx J(\mathbf{e}, \mathbf{\Phi}) \cdot \Delta \mathbf{\Phi} \\ &\approx \mathbf{J} \cdot \Delta \mathbf{\Phi}. \end{aligned} \tag{16}$$

Equation 16 describes an incremental forward-kinematics computation. To compute an in incremental inverse kinematics, we need to solve Equation 16 for $\Delta\mathbf{\Phi}$ as follows:

$$\Delta \mathbf{e} \approx \mathbf{J} \cdot \Delta \mathbf{\Phi} \tag{17}$$

$$\Delta \mathbf{\Phi} \approx \mathbf{J}^{-1} \cdot \Delta \mathbf{e}. \tag{18}$$

Often, the Jacobian matrix is not invertible. In fact, this is the case of the Jacobian matrix of the robot arm in this assignment. Thus, instead of using the inverse of the Jacobian, we will calculate its pseudo-inverse. We make this change explicit by re-writing Equation 18 as:

$$\Delta \mathbf{\Phi} \approx \mathbf{J}^{\dagger} \cdot \Delta \mathbf{e}, \tag{19}$$

where:

$$\mathbf{J}^{\dagger} = V D^{+} U^{\mathsf{T}} \tag{20}$$

is the Moore-Penrose pseudo inverse of the Jacobian. Here, matrices $U$, $D$, and $V$ are obtained from the Singular Value Decomposition of $J$, i.e., $J = UDV^{\mathsf{T}}$.

Thus, given some desired incremental change in the end-effector position, we use (19) to compute an appropriate incremental change in joint-angle configuration.

### 3.4.3 Computing the value of $\Delta\mathbf{e}$

Equation 18 depends on the value of $\Delta\mathbf{e}$ that will take the end-effector closer to the goal location. Calculating this value is the next step of our algorithm. Let $\mathbf{\Phi}$ be the current pose of the robot arm and $\mathbf{e}$ be the current end-effector position. Also, let $\mathbf{g}$ be the goal position that we want the end effector to reach. Since we will solve the inverse kinematics using an iterative approach (i.e., gradient descent), we want to choose a value for $\Delta\mathbf{e}$ that will move $\mathbf{e}$ closer to $\mathbf{g}$. A reasonable place to start is by setting:

$$\Delta\mathbf{e} = \mathbf{g} - \mathbf{e} \tag{21}$$

As in the derivation of the gradient descent for scalar functions, non-linearity prevents the exact solution that would satisfy Equation 21. Instead, we will take small steps, i.e.:

$$\Delta\mathbf{e} = \beta\left(\mathbf{g} - \mathbf{e}\right), \tag{22}$$

with $\beta \in [0, 1]$. The basic Jacobian inverse-kinematics algorithm is listed in Algorithm 2. The diagram in Figure 7 shows an analogy between the Jacobian method in Algorithm 2 and the gradient-descent method for a scalar function of a scalar variable.

**Algorithm 2** Inverse Kinematics Gradient descent (Jacobian)

$d \leftarrow$ acceptable distance from the goal
$\mathbf{\Phi}_i \leftarrow$ current pose
$\mathbf{e}_i \leftarrow$ current end-effector position
**while** $\|\mathbf{e}_i - \mathbf{g}\| > d$ **do**
  $\mathbf{J} \leftarrow J(\mathbf{e}_i, \mathbf{\Phi}_i)$ ▷ Compute the Jacobian for the current pose (Eq. 15)
  $\mathbf{J}_{\text{inv}} \leftarrow \mathbf{J}^{\dagger}$ ▷ Invert the Jacobian matrix (Eq. 20)
  $\Delta\mathbf{e} \leftarrow \beta(\mathbf{g} - \mathbf{e}_i)$ ▷ Compute the step to take (Eq. 22)
  $\Delta\mathbf{\Phi} \leftarrow \mathbf{J}_{\text{inv}} \cdot \Delta\mathbf{e}$ ▷ Compute the resulting change in pose (Eq. 19)
  $\mathbf{\Phi}_{i+1} \leftarrow \mathbf{\Phi}_i + \Delta\mathbf{\Phi}$ ▷ Apply change to joint angles
  $\mathbf{e}_{i+1} \leftarrow \mathbf{e}(\mathbf{\Phi}_{i+1})$ ▷ Compute new end-effector position $\mathbf{e}$ (Eq. 1)
**end while**

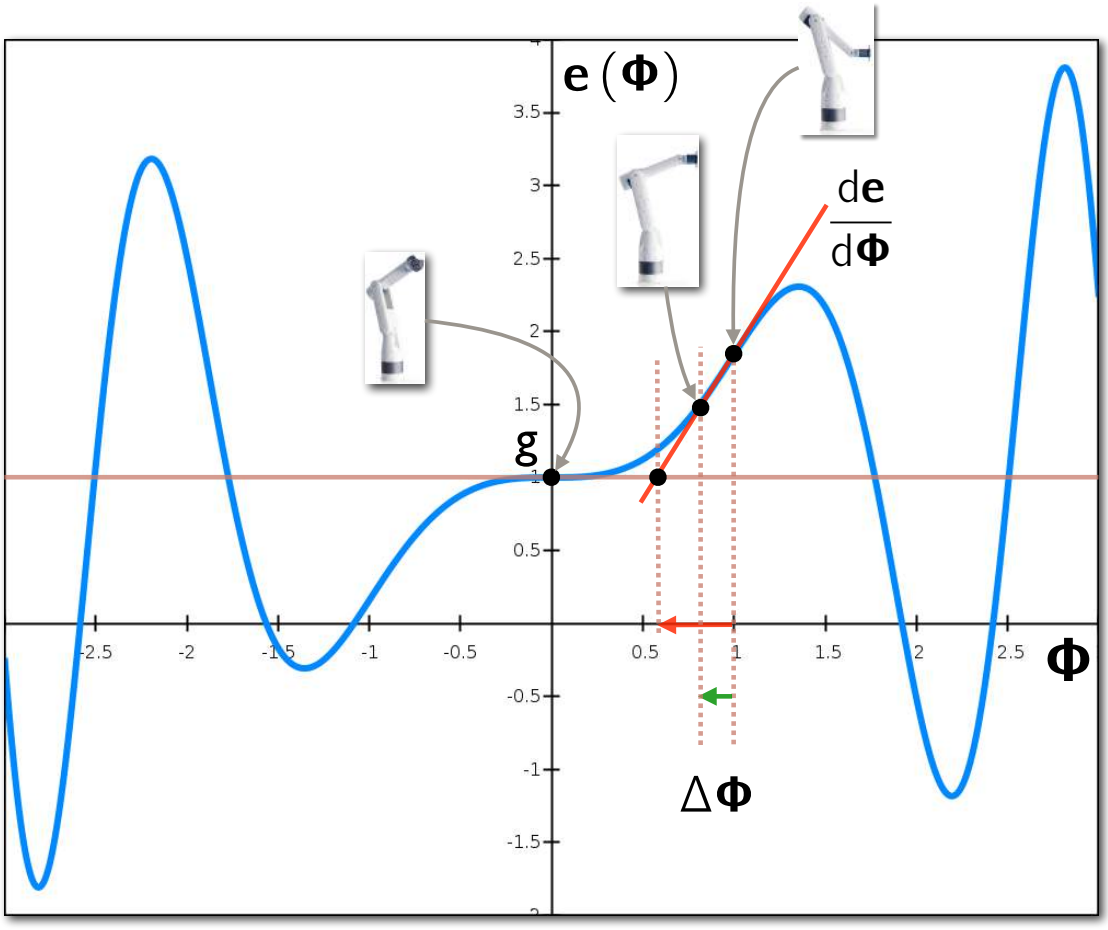



Figure 7: Jacobian Inverse Kinematics.

# A Rigid-body transformation

Rotations about x-axis, y-axis, and z-axis:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{23}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \tag{24}$$

$$R_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{25}$$

Translation by a vector $\mathbf{t} = (t_x, t_y, t_z)^{\mathsf{T}}$. The rigid-body transformation in block-matrix notation:

$$T = \begin{bmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{26}$$

Example transformation with rotation about the z-axis:

$$T = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & t_x \\ \sin\phi & \cos\phi & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

# B Calculus review

## B.1 Derivative of a scalar function of a single scalar variable

Let $f$ be a scalar function of a single variable $x$. The derivative of the function w.r.t. $x$ is:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{28}$$

The derivative of a scalar function describes the slope (i.e., rate of change) of the function at a given point $x$. Figure 8 shows the derivative of the function $f(x) = x \sin x^2 + 1$ at $x = -1$, $-0.5$, 1, 0.5, and 1. The derivative of the function is $\frac{df}{dx}(x) = 2x^2 \cos x^2 + \sin x^2$.

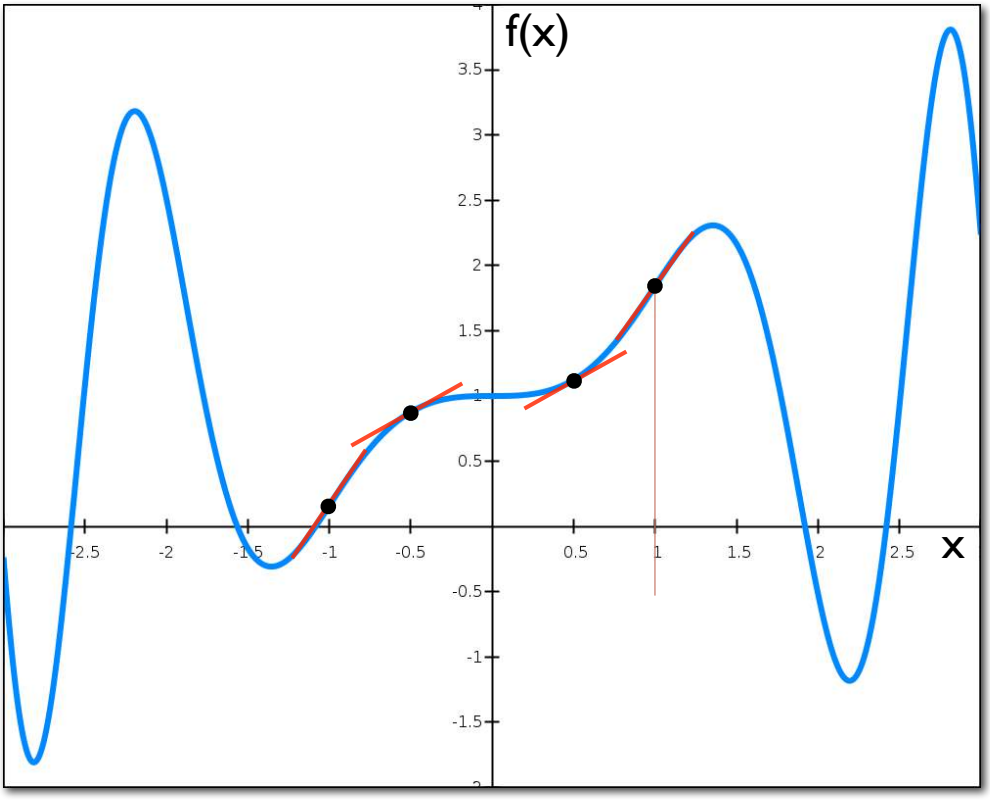



Figure 8: The derivative of the function $f(x) = x \sin x^2 + 1$ at $x = -1$, $-0.5$, 1, 0.5, and 1.

## B.2 Derivative approximation

Often, calculating the derivative analytically can be very hard. In these cases, we can make progress by approximating the derivative numerically by using discrete differences. In fact, as

long as we can evaluate the function, we can always approximate the derivative, i.e.:

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{\Delta f}{\Delta x} = \frac{f\left(x+\Delta x\right) - f\left(x\right)}{\Delta x}, \tag{29}$$

for a small $\Delta x$. Figure 9 shows the approximate derivative of the function for a small $\Delta x$.

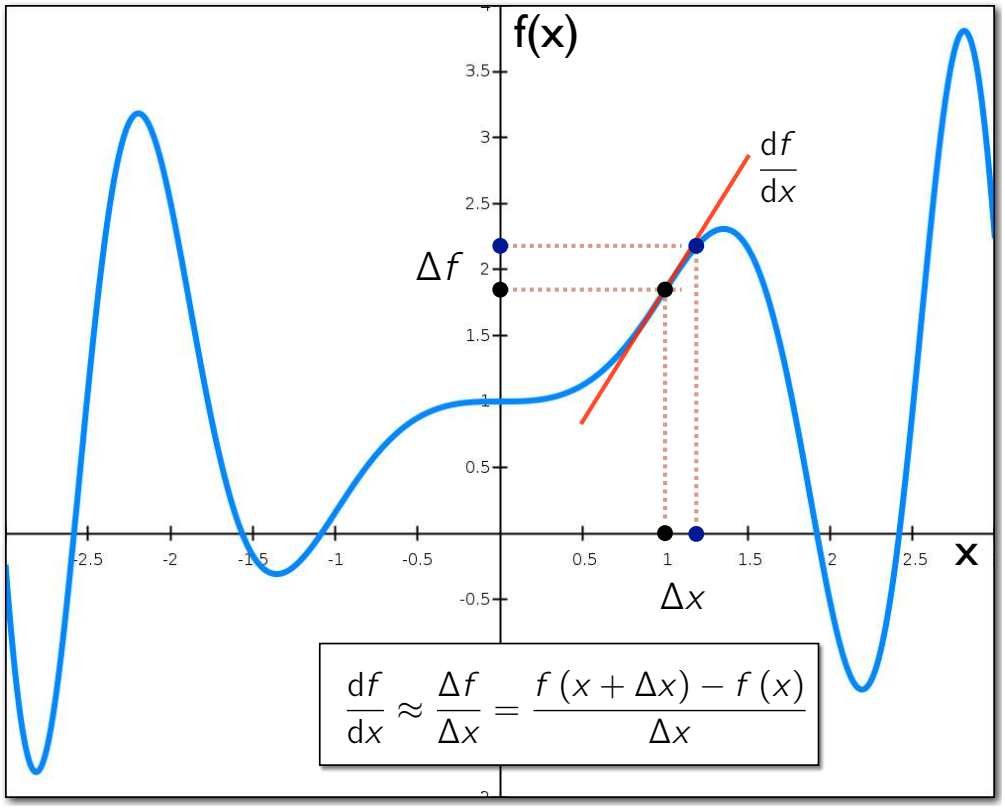



Figure 9: The approximate derivative of the function $f(x) = x \sin x^2 + 1$ at $x = -1$.

## B.3 Calculating the value of functions at nearby points

Another useful tool from derivatives is that it allows us to calculate the value of a function at nearby points. Given the value of a function at a point, $f(x)$, and its derivative, $df/dx$, we can estimate the value of the function at a point near $x$, i.e.:

$$\begin{aligned}
\frac{\Delta f}{\Delta x} &\approx \frac{\mathrm{d}f}{\mathrm{d}x} \\
\Delta f &\approx \Delta x \frac{\mathrm{d}f}{\mathrm{d}x} \\
f\left(x+\Delta x\right) - f\left(x\right) &\approx \Delta x \frac{\mathrm{d}f}{\mathrm{d}x} \\
f\left(x+\Delta x\right) &\approx f\left(x\right) + \Delta x \frac{\mathrm{d}f}{\mathrm{d}x}.
\end{aligned} \tag{30}$$

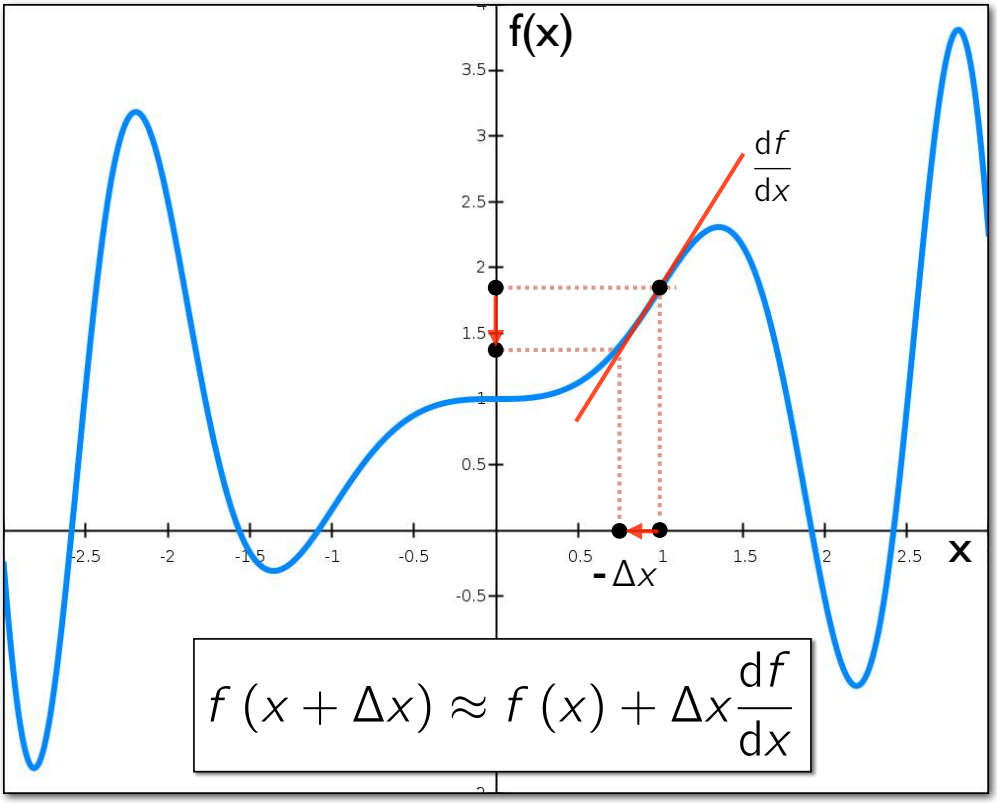



Figure 10: Derivatives allow us to calculate the value of a function at nearby points. Given the value of a function at a point, $f(x)$, and its derivative, $df/dx$, we can estimate the value of the function at a point near $x$

## B.4 The gradient-descent method

Many times, we want to find the value for which a function is zero, i.e., we want to find solutions to the equation $f(x) = 0$. This equation can be solved analytically or numerically. One of the many numerical methods to solving this equation is the gradient-descent method.

If we can evaluate $f(x)$ and $df/dx$ for any value of $x$, we can always follow the slope (i.e., gradient) in the direction towards 0. Starting at some initial value $x_0$, take small steps until we find a value $x_n$ for which $f(x_n) = 0$.

$$x_{i+1} = x_i + \Delta x. \tag{31}$$

For each step, we choose a value of $\Delta x$ that brings us closer to our goal. We can try to

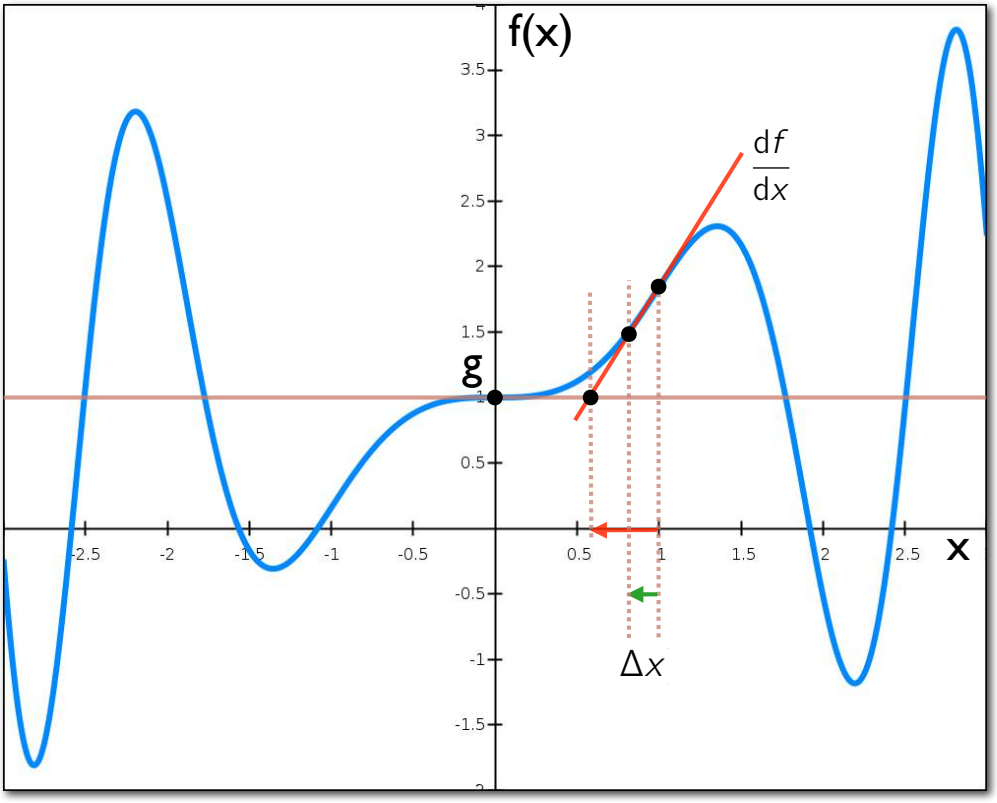



Figure 11: Computing the gradient descent's $\Delta x$ step. The red arrow shows a step that was computed using $\Delta x = (g - f(x_i)) \left(\frac{df}{dx}\right)^{-1}$, which may be too large (and over optimistic) given the function's linear approximation $df/dx$. In contrast, the green arrow shows a (smaller) step that was computed using $\Delta x = \beta (g - f(x_i)) \left(\frac{df}{dx}\right)^{-1}$, with $0 < \beta \leq 1$. For instance, we can set $\beta = 0.1$.

choose $\Delta x$ to bring us closer to where the slope passes through 0, i.e.:

$$\begin{aligned}
\frac{\Delta f}{\Delta x} &\approx \frac{df}{dx} \\
\Delta f &\approx \Delta x \frac{df}{dx} \\
\cancelto{0}{f(x_i + \Delta x)} - f(x_i) &\approx \Delta x \frac{df}{dx} \\
-f(x_i) &\approx \Delta x \frac{df}{dx} \\
\Delta x &= -f(x_i) \left(\frac{df}{dx}\right)^{-1}. \qquad (32)
\end{aligned}$$

If we want to find where a function equals some value $g$ instead of 0, we can think of the

problem as minimizing $f(x) - g$ and just step towards $g$, i.e.:

$$\Delta x = (g - f(x_i)) \left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)^{-1}. \tag{33}$$

However, Equation 33 assumes that our linear approximation of the function given by its derivative is reliable for large values of $\Delta x$. However, this is not the case for non-smooth functions with varying derivatives. A safer way to choosing $\Delta x$ is to multiply it by a parameter $\beta \in (0, 1]$ to scale the step. With the inclusion of the $\beta$ scale factor, Equation 33 can be re-written as:

$$\Delta x = \beta (g - f(x_i)) \left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)^{-1}. \tag{34}$$

Figure 11 shows the $\Delta x$ steps computed by using Equations 33 and 34.

**Algorithm 3** Gradient descent (scalar function of a single scalar variable)

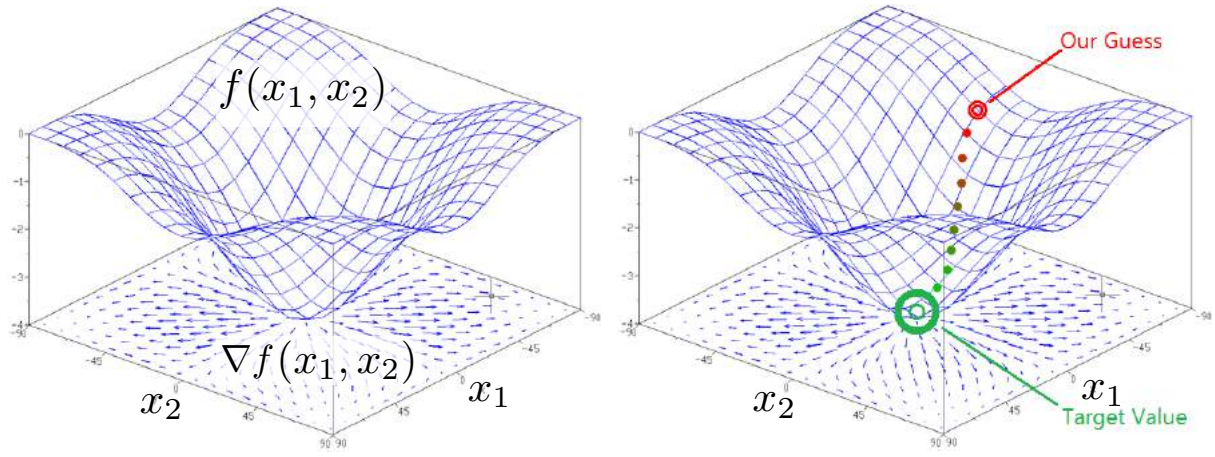
$x_0 \leftarrow$ starting value
$f_0 \leftarrow f(x_0)$ ▷ Evaluate $f$ at $x_0$
**while** $f_n \neq g$ **do**
  $s_i \leftarrow \frac{\mathrm{d}f}{\mathrm{d}x}(x_i)$ ▷ Compute slope
  $x_{i+1} \leftarrow x_i + \beta (g - f_i) \frac{1}{s_i}$ ▷ Take a step along $\Delta x$
  $f_{i+1} \leftarrow f(x_{i+1})$ ▷ Evaluate $f$ at new $x_{i+1}$
**end while**

## B.5 Derivative of a scalar function of multiple scalar variables (i.e., vector variable)

Let $f$ be a scalar function of a vector variable. The vector variable is $\mathbf{x} = (x_1, x_2, \ldots, x_N)^{\mathsf{T}}$. This type of function is also called a scalar function of multiple variables or multi-variate function. The value of the function at a point $\mathbf{x}$ is given by $f(\mathbf{x})$ or $f(x_1, x_2, \ldots, x_N)$, and its derivative w.r.t. $x$ is:

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}} = \nabla f = \frac{\partial f}{\partial (x_1, x_2, \ldots, x_N)} = \left[\frac{\partial f}{\partial x_1} \; \frac{\partial f}{\partial x_2} \; \cdots \; \frac{\partial f}{\partial x_N}\right]^{\mathsf{T}}, \tag{35}$$

which is called the *Gradient* of $f$ at $x$, and denoted by $\nabla f$. An example of a gradient vector field of a function $f(x_1, x_2)$ is shown in Figure 12.




(a) (b)

Figure 12: (a) Gradient vector field of two-variate function. (b) Gradient-descent method. In the gradient-descent method, we want to take the direction inverse to the gradient as the gradient (slope) points "uphill". Plots adapted from `https://goo.gl/zejgBw`.

## B.6 Derivative of a vector function of a single scalar variable

Let $\mathbf{r}$ be a vector function representing the 3-D motion of a particle as a function of time $t$:

$$\mathbf{r} = [r_x \; r_y \; r_z]^\mathsf{T} . \tag{36}$$

Its derivative is given by:

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \left[\frac{\mathrm{d}r_x}{\mathrm{d}t} \; \frac{\mathrm{d}r_y}{\mathrm{d}t} \; \frac{\mathrm{d}r_z}{\mathrm{d}t}\right]^\mathsf{T} , \tag{37}$$

which is also a vector representing the velocity of the particle at time $t$.

## B.7 Derivative of a vector function of a vector variable

Some applications require us to calculate derivatives of vector quantifies with respect to other vector quantities. For example, if $\mathbf{f}$ is a vector-valued function of a vector of variables, $\mathbf{x}$. Here, $\mathbf{f}(\mathbf{x}) = (f_1, f_2, \ldots, f_M)^\mathsf{T}$ and $\mathbf{x} = (x_1, x_2, \ldots, x_N)^\mathsf{T}$.

The derivative of $\mathbf{f}$ w.r.t. $\mathbf{x}$ is:

$$\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{x}} = J(\mathbf{f}, \mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial x_1} & \frac{\partial f_M}{\partial x_2} & \cdots & \frac{\partial f_M}{\partial x_N}, \end{bmatrix} \tag{38}$$

and is called the *Jacobian*. By inspecting the Jacobian matrix, we notice that it can be written as a stack of gradients $\nabla f_i$ as rows, i.e.:

$$J(\mathbf{f}, \mathbf{x}) = \begin{bmatrix} \nabla f_1 \\ \nabla f_2 \\ \vdots \\ \nabla f_M \end{bmatrix} \tag{39}$$

or as matrix of columns where each column is derivative of the vector function with respect to a component of the vector variable, i.e.:

$$J(\mathbf{f}, \mathbf{x}) = \begin{bmatrix} \frac{\partial \mathrm{f}}{\partial x_1} & \frac{\partial \mathrm{f}}{\partial x_2} & \cdots & \frac{\partial \mathrm{f}}{\partial x_N} \end{bmatrix} \tag{40}$$

# References

[1] David E. Breen. Cost minimization for animated geometric models in computer graphics. *The Journal of Visualization and Computer Animation*, 8(4):201–220, 1997.