ROB 101: Computational Linear Algebra

Noah Peters

March 31, 2023

Abstract

Lecture notes for ROB 101 at the University of Michigan. LaTeXtemplate by Pingbang Hu.

Contents

Chapter 11

Solutions of Nonlinear Equations

Lecture 18: Review and Approximating Nonlinear Equations

14 Mar. 9:00

11.1 Bisection Algorithm

Theorem 11.1.1 (Intermediate Value Theorem)**.** Suppose $f : \mathbb{R} \to \mathbb{R}$ is a continuous function and you know two real numbers, $a < b$, such that $f(a) \cdot f(b) < 0$. Then $\exists c \in \mathbb{R}$ such that:

$$a < c < b$$
$$f(c) = 0$$

Using the midpoint between two numbers to find the root may not give us the root right away. Further, the root isn't always exactly in between a and b. So, we can use the **bisection algorithm** to *approximate* roots:

Algorithm 11.1: Bisection Algorithm

Data: $a < b \in \mathbb{R}$ such that $f(a) \cdot f(b) < 0$

1 $c = (a+b)/2$ **if** $f(c) = 0$ **then**
2   **return** $x^* = c$
3 **else**
4   **if** $f(c) \cdot f(a) < 0$ **then**
5         $b = c$
6   **else**
7         $a = c$
8 Loop back to **1**.

11.2 Derivatives and Approximation

Definition 11.2.1 (Derivative)**.** A **derivative** is the slope of a function at a specific point.

There are 3 ways to represent the numerical approximation of a derivative:

1. **Forward Difference Approximation**: $\frac{df(x_0)}{dx} = \frac{f(x_0+h)-f(x_0)}{h}$
2. **Backward Difference Approximation**: $\frac{df(x_0)}{dx} = \frac{f(x_0)-f(x_0-h)}{h}$
3. **Symmetric Difference Approximation**: $\frac{df(x_0)}{dx} = \frac{f(x_0+h)-f(x_0-h)}{2h}$

Remark. If the 3 approximations above don't agree, then the limit does not exist and the function is not differentiable.

For a differentiable function, $f(x)$, $f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0)$ near point $(x_0, f(x_0))\ \forall x_0$. We can use this idea to *find roots* using linear approximations to nonlinear functions.

11.3 Newton's Method

To find the roots using linear approximations to nonlinear functions we can use **Newton's Method**:

Definition 11.3.1 (Newton's Method)**.** Assume $f : \mathbb{R} \to \mathbb{R}$ is differentiable everywhere. Let x_k be our current estimate of a root, then:

$$f(x) \approx f(x_k) + \frac{df(x_k)}{dx}(x - x_k).$$

We want x_{k+1} such that $f(x_{k+1}) = 0$.

If we solve for x_{k+1} we get:

$$x_{k+1} = x_k - \frac{df(x_k)}{dx}^{-1} f(x_k).$$

This method takes very big "steps", so it may be more beneficial to take smaller "steps". This leads to the **damped Newton Method**,

$$x_{k+1} = x_k - \epsilon\big(\frac{df(x_k)}{dx}\big)^{-1} f(x_k),$$

where $0 < \epsilon < 1$ A typical value may be $\epsilon = 0.1$.

Lecture 19: Vectors and Approximating Nonlinear Equations

11.4 Vectors and Nonlinear Equations

16 Mar. 9:00

We can use vectors for linear approximations by understanding partial derivatives. Given nonlinear funcitons $f : \mathbb{R}^m \to \mathbb{R}^m$ we want the linear approximation at $x_0 \in \mathbb{R}^m$.

$$f(x) \approx f(x_0) + A(x - x_0)$$

where $A_{n\times m}, x, x_0 \in \mathbb{R}^m$, and $f(x_0), f(x) \in \mathbb{R}^n$. Here, A represents a matrix made up of partial derivatives.

Remark. Everything is the same as finding a linear approximation at a point. We are just replacing the slope with a matrix and the xs with vectors.

11.5 Partial Derivatives

Set $x = x_0 + he_i$ where he_i is some small outside adjustment to x_0 such that all x_0 remain the same except the i-th component.

$$x_0 + he_i = \begin{bmatrix} x_{01} \\ x_{02} \\ \vdots \\ x_{0m} \end{bmatrix} + h \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} x_{01} \\ x_{02} \\ \vdots \\ x_{0i} + h \\ \vdots \\ x_{0m} \end{bmatrix} \text{ where } h \text{ is small}$$

Equivalently, we have:

$$f(x_0 + he_i) \approx f(x_0) + A(x_0 + he_i - x_0) = f(x_0) + ha_i^{col}$$

where we can now solve for a_i^{col}, which represents the derivative of f with respect to x_i.

We can represent the numerical approximation of a partial derivative similar to how we represented the numerical approximation of a standard derivative. A partial derivative is represented with the mathematical symbol del: ∂.

11.6 Jacobian

Given the nonlinear functions $f : \mathbb{R}^m \to \mathbb{R}^m$, the **Jacobian** is

$$\frac{\partial f(x)}{\partial x} := \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \frac{\partial f(x)}{\partial x_2} \cdots \frac{\partial f(x)}{\partial x_m} \end{bmatrix}_{n \times m}.$$

- The partial derivatives are stacked to form a matrix
- For each $x \in \mathbb{R}^m$, $\frac{\partial f(x)}{\partial x}$ is an $n \times m$ matrix
- The **gradient** of $f : \mathbb{R}^m \to \mathbb{R}$ is a special Jacobian that for each $x \in \mathbb{R}^m$, $\nabla f(x)$ is a $1 \times m$ matrix

$f : \mathbb{R}^m \to \mathbb{R}^n$ looks like:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

$\frac{\partial f(x)}{\partial x}$ looks like:

$$\frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_m} \end{bmatrix}$$

The linear approximation of nonlinear functions $f : \mathbb{R}^m \to \mathbb{R}^n$ at point $x_0 \in \mathbb{R}^m$ is

$$f(x) \approx f(x_0) + A(x - x_0) = f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0).$$

Problem 11.6.1. We have two functions:

$$f_1(x_1, x_2) = \log(x_1) + \sqrt{x_2}$$
$$f_2(x_1, x_2) = x_1 \cdot x_2$$

and let $x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Answer. $f : \mathbb{R}^2 \to \mathbb{R}^2$ so $\frac{\partial f(x)}{\partial x}$ is a 2×2 matrix. Using forward approximation we have $\frac{\partial f(x_0)}{\partial x} = \frac{f(x_0 + he_i) - f(x_0)}{h}$ with $h = 0.001$.

- f_1 w.r.t x_1:

$$\frac{\partial f_1(x_1, x_2)}{\partial x_1} = \frac{(\log(1 + 0.001) + \sqrt{2}) - (\log(1) + \sqrt{2})}{0.001} = 0.43408$$

- f_1 w.r.t. x_2:

$$\frac{\partial f_1(x_1, x_2)}{\partial x_2} = \frac{(\log(1) + \sqrt{2 + 0.0001}) - (\log(1) + \sqrt{2})}{0.001} = 0.35351$$

- f_2 w.r.t. x_1:

$$\frac{\partial f_1(x_1, x_2)}{\partial x_1} = \frac{(1.001)(2) - (1)(2)}{0.001} = 2$$

- f_2 w.r.t. x_2:

$$\frac{\partial f_1(x_1, x_2)}{\partial x_2} = \frac{(1)(2.001) - (1)(2)}{0.001} = 1$$

So, the linear approximation is $f(x) \approx f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0)$

$$\begin{aligned} f(x) &\approx f(x_0) + \frac{\partial f(x_0)}{\partial x}(x - x_0) \\ &= \begin{bmatrix} \log(1) + \sqrt{2} \\ (1)(2) \end{bmatrix} + \begin{bmatrix} 0.43408 & 0.35351 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 - 1 \\ x_2 - 2 \end{bmatrix} \end{aligned}$$

⊛

11.7 Newton-Raphson Method

Just as Newton's Method was a useful tool for linear approximations, we can use the **Newton-Raphson Method** for *non-linear approximations.* Given $f : \mathbb{R}^n \to \mathbb{R}^n$, we want to find a root $f(x_0) = 0$. Using our approximation above, we can substitute in x_{k+1} and solve for it:

$$x_{k+1} = x_k - \frac{\partial f(x)}{\partial x}^{-1} f(x_k)$$

Remark. Similarly to when we were solving $Ax - b = 0$ we made sure $\det(A) \neq 0$, we want to make sure $\det\left(\frac{\partial f(x_k)}{\partial x} \neq 0\right)$.

We can find Δx_k to avoid the inverses in the equations above:

$$\Delta x_k = -\left(\frac{\partial f(x_k)}{\partial x}\right)^{-1} f(x_k)$$

Instead of the inverses or dividing matrices, we solve for Δx_k using LU or QR factorization. This can be done using the Newton-Raphson algorithm:

Algorithm 11.2: Newton-Raphson Algorithm

Data: f

1 $F \leftarrow$ **LU** $\left(\frac{\partial f(x_k)}{\partial x}\right)$ `// find` Δx_k
2 $y \leftarrow$ **ForwardSub** $(F.L, F.P \cdot -f(x_k))$
3 $\Delta x_k \leftarrow$ **BackwardSub** $(F.U, y)$
4 $x_{k+1} \leftarrow x_k + \Delta x_k$ `// use` Δx_k `to find` x_{k+1}
5 **if** $f(x_{k+1}) = 0$ **then**
6   **return** x_{k+1} *as the root*
7 **else**
8   loop back to **1**.

If we replace $x_{k+1} = x_k \Delta x_k$ with

$$x_{k+1} = x_k + \varepsilon \Delta x_k$$

we get the **Damped Newton-Raphson Method** for $\varepsilon > 0$ (usually $\varepsilon = 0.1$ is sufficient). This prevents Δx_k from being too big by decreasing the step size.

Chapter 12

Basic Ideas of Optimization

Lecture 20: Gradient Descent

21 Mar. 9:00

12.1 Gradient

Let $f : \mathbb{R}^m \to \mathbb{R}$, then the **gradient** of f is the partial derivatives of f with respect to x_i:

Definition 12.1.1 (Gradient (∇)).

$$\nabla f(x_0) = \left[\frac{\partial f(x_0)}{\partial x_1} \frac{\partial f(x_0)}{\partial x_2} \dots \frac{\partial f(x_0)}{\partial x_m}\right]_{1\times m}.$$

For linear approximation about a point:

$$f(x) \approx f(x_0) + \nabla f(x_0)(x - x_0)$$

12.2 Building Towards Optimization

Optimization is finding a potential set of solutions to a problem in $\mathbb{R}^m$ The **cost function** $f : \mathbb{R}^m \to \mathbb{R}$ allows us to compare elements of $\mathbb{R}^m$ in order for us to decide which are more advantageous to us.

1. **REGRET** functions minimize. If $*$ is the minimum point of interest, as $x \in \mathbb{R}^m$ gets close to our x^*, it is small and as x get far from x^*, it is large. Mathematically:

$$x^* = \mathbf{argmin}_{x\in\mathbb{R}^m} f(x).$$

2. **REWARD** functions maximize. Here we will focus on minimization.

Suppose we start at $x_k \in \mathbb{R}$ and we want to find $x_{k+1} \in \mathbb{R}$ where $f(x_{k+1}) < f(x_k)$. We note that $f(x_{k+1}) - f(x_k) < 0$ if and only if $\frac{\partial f(x_k)}{\partial x}(x_{k+1} - x_k) < 0$.

Remark. Make sure that you do not begin with $\frac{\partial f(x_k)}{\partial x} = 0$, as this would indicate x_k is already an extremum.

So, we let $\Delta x_k = -s\frac{\partial f(x_k)}{\partial x}$ for $s > 0$ (**step size**). If s is too big then we may overshoot our estimate. We typically use $s \approx 0.1$. Solving for x_{k+1} (i.e. our *next best guess* closer to the local extremum):

$$x_{k+1} = x_k - s\frac{\partial f(x_k)}{\partial x}$$

12.3 Gradient Descent

Note that the gradient vanishes at local minima: $\nabla f(x^*) = 0$. In order to find an extremum (in our case: a minimum) we can start at some arbitrary x_k and calculate (for a satisfactory k), $x_{k+1} = x_k - s(\nabla f(x_k))^T$.

Remark. We transpose $\nabla f(x_k)$ because the gradient is a row vector, so we must transpose it into a column.

Finding the Minimum: Gradient Descent Algorithm

Note that our *key condition* is

$$\nabla f(x_k)\Delta x_k < 0.$$

Using the above, we can find a minimum starting with the linear approximation of $f : \mathbb{R}^m \to \mathbb{R}$ near x_k:

$$f(x) \approx f(x_k) + \nabla f(x_k)(x - x_k)$$

We want x_{k+1} such that $f(x_{k+1}) < f(x_k)$. We find that if $\Delta x_k = -s(\nabla f(x_k))^T$ then we have:

$$\nabla f(x_k)\Delta x_k = -s\|(\nabla f(x_k))^T\|^2 < 0,\ \forall s > 0$$

We can construct an algorithm from this:

Algorithm 12.1: Gradient Descent Algorithm

Data: $f, x_i \leftarrow 0, s \leftarrow 0.1$

while $\|\nabla f(x_i)\| < tol \ \ \& \ \ i < i_{\max}$ **do**
- $\Delta x_i \leftarrow -s \cdot \nabla f(x_i)$
- $x_i \leftarrow x_i + \Delta x_i$
- $i \mathrel{+}= 1$

return x_i

Lecture 21: Root Finding With the Second Derivative

12.4 Hessian

23 Mar. 9:00

Definition 12.4.1. The **Hessian** of a function $f : \mathbb{R}^m \to \mathbb{R}$ is the Jacobian of the *gradient transpose* of f:

$$\nabla^2 f(x) \coloneqq \frac{\partial(\nabla f(x))^T}{\partial x}$$

where $x \in \mathbb{R}^m$ and $f(x) \in \mathbb{R}$.

Definition 12.4.2 (Gradient Transpose)**.** The **gradient transpose** is defined as:

$$(\nabla f(x))^T \coloneqq \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_m} \end{bmatrix}_{m\times 1}$$

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_m^2} \end{bmatrix}$$

Figure 12.1: The Hessian in terms of individual entries.

The **Jacobian** for $g : \mathbb{R}^m \to \mathbb{R}^m$ is

$$\frac{\partial g(x)}{\partial x} := \left[\frac{\partial g(x)}{\partial x_1} \dots \frac{\partial g(x)}{\partial x_m}\right]_{m \times m}.$$

To summarize the above, the two methods for finding a minimum that we have seen include:

Definition 12.4.3 (*Scalar* Optimization (Newton's Method)). For $f : \mathbb{R} \to \mathbb{R}$:

$$x_{k+1} = x_k - \left(\frac{\partial^2 f(x_k)}{\partial x^2}\right)^{-1} \frac{\partial f(x_k)}{\partial x}$$

and its *damped* version where $0 < s < 1$:

$$x_{k+1} = x_k - s(\left(\frac{\partial^2 f(x_k)}{\partial x^2}\right)^{-1} \frac{\partial f(x_k)}{\partial x}).$$

Definition 12.4.4 (*Vector* Optimization (Newton-Raphson)). For $f : \mathbb{R}^m \to \mathbb{R}$:

$$\nabla^2 f(x_k) \Delta x_k = -(\nabla f(x_k))^T$$

and its *damped* version where $0 < s < 1$:

$$x_{k+1} = x_k s \Delta x_k.$$

Remark. Use LU or QR factorization (i.e. $Ax = b$) to solve for Δx_k.

The Hessian used in the Newton-Raphson algorithm gives us the root of the gradient function. For us, our goal was to find the local minimum. In some problems, Newton-Raphson with the Hessian has a faster

Appendix