Christopher Lum
lum@uw.edu

Lecture 08b
Numerically Calculating Partial Derivatives

The YouTube video entitled 'Numerically Calculating Partial Derivatives' that covers this lecture is located at https://youtu.be/G2gxvRjQHxc.

Outline

-Partial Derivatives
- -1 Variable
- -2 Variables
- -n Variables

References

-https://en.wikipedia.org/wiki/Numerical_differentiation

Partial Derivatives

Consider a function of n variables

$$f(x_1, x_2, \ldots, x_n) = f(\overline{x}) \qquad \textbf{(Eq.1)}$$

where $\overline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$

We can calculate the gradient f w.r.t. the independent variables

$$\nabla f(\overline{x}) = \frac{\partial}{\partial \overline{x}}[f(\overline{x})] = \frac{\partial f}{\partial \overline{x}} = \begin{pmatrix} \frac{\partial f(\overline{x})}{\partial x_1} \\ \frac{\partial f(\overline{x})}{\partial x_2} \\ \cdots \\ \frac{\partial f(\overline{x})}{\partial x_n} \end{pmatrix} \qquad \textbf{(Eq.2)}$$

As expected, the gradient is a function of the point where it is evaluated

$$\nabla f(\overline{x}_o) = \frac{\partial}{\partial \overline{x}}[f(\overline{x})] \mid_{\overline{x}=\overline{x}_o} = \frac{\partial f(\overline{x}_o)}{\partial \overline{x}} = \begin{pmatrix} \frac{\partial f(\overline{x}_o)}{\partial x_1} \\ \frac{\partial f(\overline{x}_o)}{\partial x_2} \\ \cdots \\ \frac{\partial f(\overline{x}_o)}{\partial x_n} \end{pmatrix} \qquad \textbf{(Eq.3)}$$

The gradient measures how the function f changes w.r.t. each independent variable x_1, x_2, ..., x_n. For example, the j^{th} entry in the gradient measures the rate of change of f if only x_j varies.

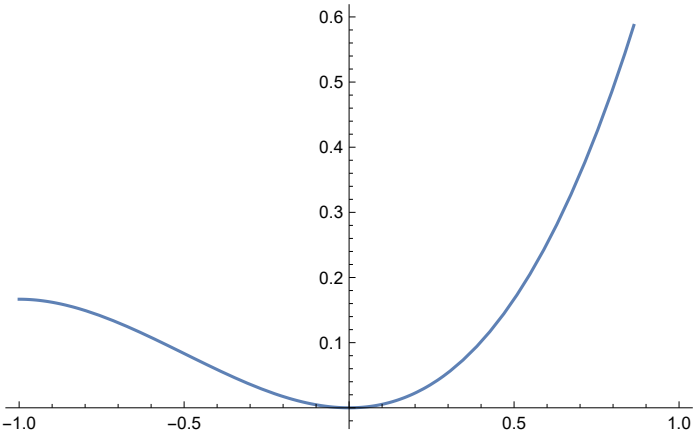
1 Variable

Example: Function of one variable

Consider a simple function of 1 variable

$$f(x_1) = \frac{1}{3} x_1^3 + \frac{1}{2} x_1^2 \qquad \textbf{(Eq.E.1)}$$

```
fA[x1_] = 1/3 x1^3 + 1/2 x1^2;
Plot[fA[x1], {x1, -1, 1}]
```



The gradient of this is the simple derivative

```
dfAdx1[x1_] = D[fA[x1], x1];
gradfA[x1_] = (dfAdx1[x1])
```

x1 + x1^2

So the gradient at $x_{1,o}$ = 1/2 is given by

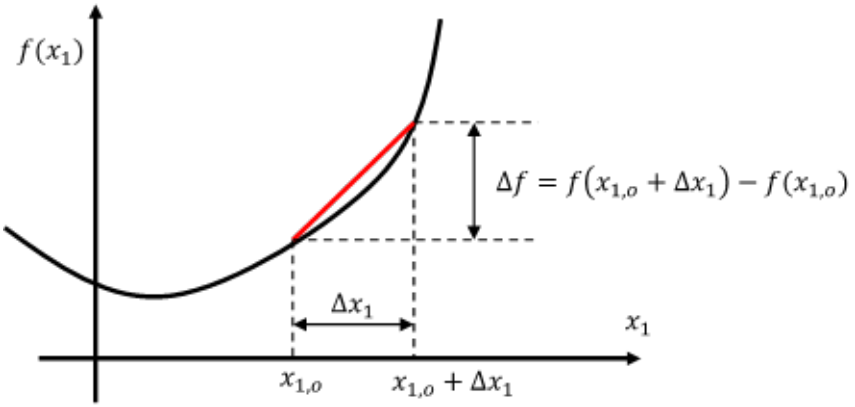
```
x1o = 1 / 2;
gradfA[x1o]
```

$\frac{3}{4}$

We can easily compute this because we can easily obtain $\nabla f\,(\overline{x}_o)$ analytically (because $f(\overline{x})$ was so simple).

Difference Quotient

If the function does not have an analytical description (for example if it is a lookup table) we may not be able to calculate analytical derivatives. Therefore, we seek a method to numerically compute this derivative. This involves numerically approximating the partial derivatives. To illustrate this, let us investigate function in 1 dimension as shown below.



As can be seen, a simple approximation of the slope of $\frac{\partial f(x_1)}{\partial x_1}\,|_{x_1=x_{1,o}}$ can be calculated using a simple rise over run formula. Namely

$$\frac{\partial f(x_1)}{\partial x_1}\,|_{x_1=x_{1,o}} \approx \frac{\text{rise}}{\text{run}} = \frac{f(x_{1,o}+\Delta x_1)-f(x_o)}{\Delta x_1} \qquad \text{(Newton quotient)} \qquad \textbf{\textit{(Eq.4)}}$$

This is the familiar, standard difference quotient (AKA the Newton quotient or Fermat's difference quotient)

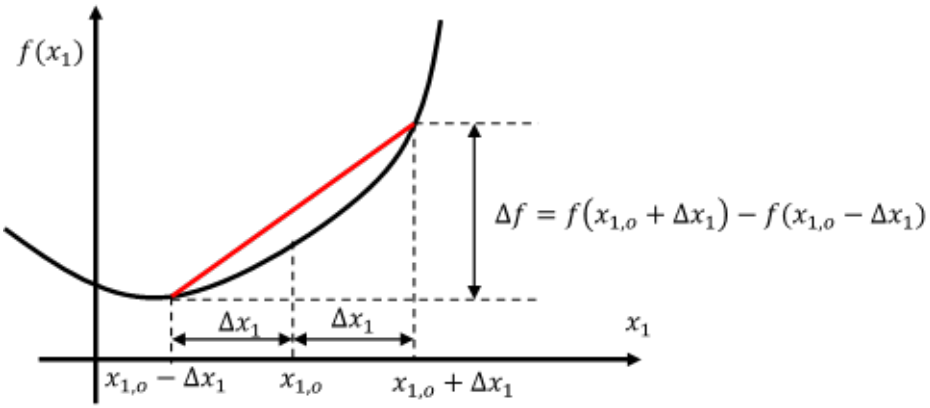
```
Δx1 = 0.1;

Print["function values"]
fA[x1o + Δx1] // N
fA[x1o] // N
Print[""]

Print["rise"]
rise = fA[x1o + Δx1] - fA[x1o]
Print[""]

Print["run"]
run = Δx1
Print[""]

Print["approximate slope"]
rise
run
Print[""]
```

```
function values

0.252

0.166667


rise

0.0853333


run

0.1


approximate slope

0.853333
```

We see this is very close to the analytical value.

Symmetric Difference Quotient

An alternative formulation involves using points to the left and right of the point of interest



This is known as the symmetric difference quotient.

$$\frac{\partial f(x_1)}{\partial x_1}\Big|_{x_1=x_{1,o}} \approx \frac{\text{rise}}{\text{run}} = \frac{f(x_{1,o}+\Delta x_1)-f(x_o-\Delta x_1)}{2\,\Delta x_1} \qquad \text{(symmetric difference quotient)} \qquad \textbf{\textit{(Eq.5)}}$$

The symmetric difference quotient is more accurate approximate than the one sided Newton quotient.

```
Print["function values"]
fA[x1o + Δx1] // N
fA[x1o - Δx1] // N
Print[""]

Print["rise"]
rise = fA[x1o + Δx1] - fA[x1o - Δx1]
Print[""]

Print["run"]
run = 2 Δx1
Print[""]

Print["approximate slope"]
rise
----
run
Print[""]
```

```
function values

0.252

0.101333


rise

0.150667
```

```
run
0.2


approximate slope
0.753333
```

The benefit of the numerical approach is that we did not need to analytically calculate the derivative. In fact, we do not even need an analytical description of f. We just need to be able to input specific values in the function and by examining how the function responds to these inputs, we can numerically approximate the slope/gradient.

2 Variables

We can extend this idea to a function of 2 variables

Example: Function of two variables

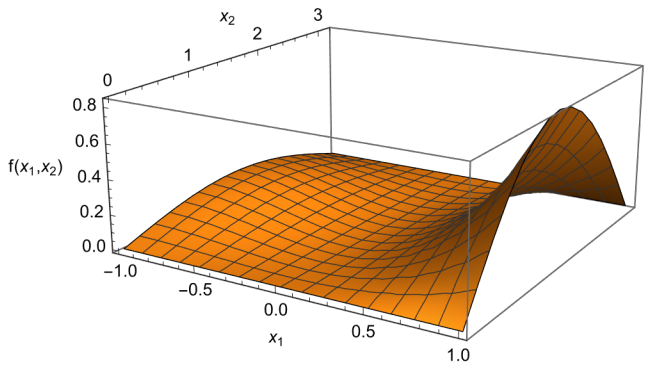
Consider a function of two variables

$$f(x_1, x_2) = \left(\frac{1}{3} {x_1}^3 + \frac{1}{2} {x_1}^2\right) \sin(x_2) \qquad \textbf{(Eq.E.1)}$$

Note that at $x_2 = \pi/2 + \pm 2\pi k$, this is the same as the 1 dimensional function we considered previously.

```
fB[x1_, x2_] = (1/3 x1^3 + 1/2 x1^2) Sin[x2]
p1 = Plot3D[fB[x1, x2], {x1, -1, 1}, {x2, 0, π},
  AxesLabel → {"x₁", "x₂", "f(x₁,x₂)"},
  PlotRange → All]
```

$$\left(\frac{x1^2}{2} + \frac{x1^3}{3}\right) \text{Sin}[x2]$$



The gradient of this is given by Eq.3

```
dfBdx1[x1_, x2_] = D[fB[x1, x2], x1];
dfBdx2[x1_, x2_] = D[fB[x1, x2], x2];

gradfB[x1_, x2_] = ( dfBdx1[x1, x2] )
                   ( dfBdx2[x1, x2] );
gradfB[x1, x2] // MatrixForm
```

$$\begin{pmatrix} (x1 + x1^2)\ \text{Sin}[x2] \\ \left(\frac{x1^2}{2} + \frac{x1^3}{3}\right)\ \text{Cos}[x2] \end{pmatrix}$$

So the gradient at $\overline{x}_o = \begin{pmatrix} 1/2 \\ 30\,\frac{\pi}{180} \end{pmatrix}$ is given by

```
x1o = 1 / 2;
x2o = 30 π / 180;

gradfB[x1o, x2o] // MatrixForm // N
```

$$\begin{pmatrix} 0.375 \\ 0.144338 \end{pmatrix}$$

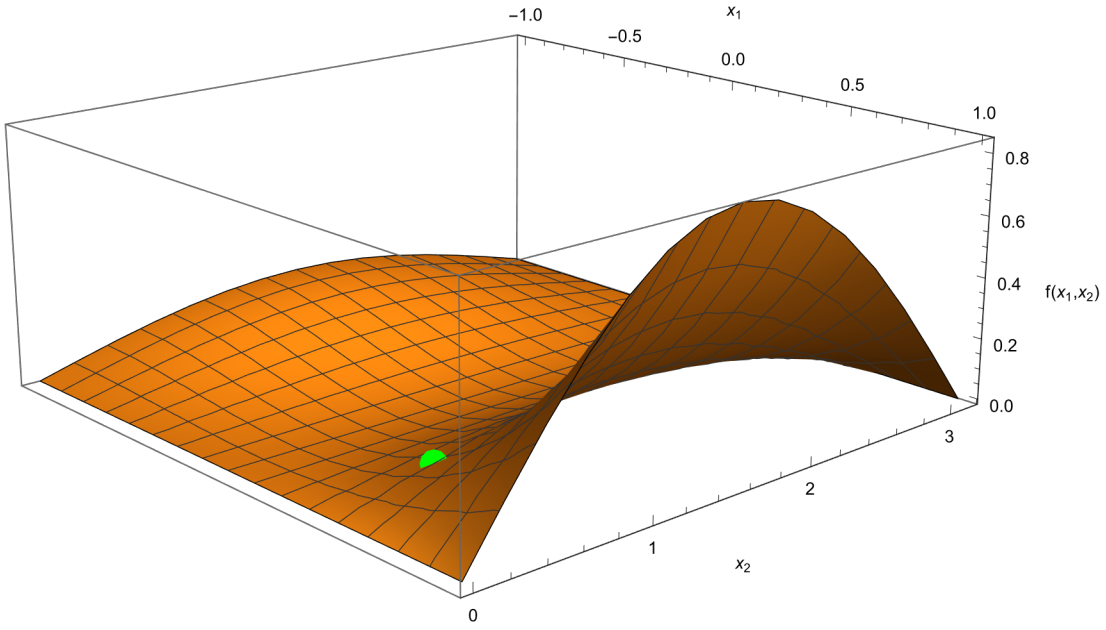
So we have

$$\nabla f(\overline{x}_o) = \begin{pmatrix} \frac{\partial f(\overline{x}_o)}{\partial x_1} \\ \frac{\partial f(\overline{x}_o)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 0.375 \\ 0.144338 \end{pmatrix}$$

Note that $\frac{\partial f(\overline{x}_o)}{\partial x_1}$ is not the same value as we computed in the 1D example due to the influence/effect of x_2.

We are effectively calculating the slope of the function f at the point $\overline{x}_o$

```
(*Define the point of interest*)
P = {x1o, x2o, fB[x1o, x2o]};

(*Plot using Graphics3D*)
p2 = Graphics3D[
    {
     AbsolutePointSize[15], Green, Point[{P[[1]], P[[2]], P[[3]]}]
    }
  ];

(*Show point on top of surface*)
Show[p1, p2]
```



We can adapt the symmetric difference quotient expression to account for 2 variables. For example, consider the term $\frac{\partial f(\overline{x}_o)}{\partial x_1}$. This is the rate of the change of the function f as <u>only</u> x_1 changes (we therefore hold all other independent variables constant). Therefore, the update to the symmetric difference quotient formula is

$$\frac{\partial f(\overline{x}_o)}{\partial x_1} \approx \frac{\text{rise}}{\text{run}} = \frac{f(\overline{x}_{1+}) - f(\overline{x}_{1-})}{2\,\Delta x_1} \qquad \textbf{(Eq.6)}$$

where $\overline{x}_{1+} = \begin{pmatrix} x_{1,o} + \Delta x_1 \\ x_{2,o} \end{pmatrix}$ (perturb from $\overline{x}_o$ only in x_1 by $+\Delta x_1$)

$\overline{x}_{1-} = \begin{pmatrix} x_{1,o} - \Delta x_1 \\ x_{2,o} \end{pmatrix}$ (perturb from $\overline{x}_o$ only in x_1 by $-\Delta x_1$)

Δx_1 = scalar perturbation distance (small number)

In a similar fashion, we can examine how the function changes as we change/perturb <u>only</u> x_2

$$\frac{\partial f(\bar{x}_o)}{\partial x_2} \approx \frac{\text{rise}}{\text{run}} = \frac{f(\bar{x}_{2+}) - f(\bar{x}_{2-})}{2\,\Delta x_2} \qquad \textbf{(Eq.6)}$$

where $\bar{x}_{2+} = \begin{pmatrix} x_{1,o} \\ x_{2,o} + \Delta x_2 \end{pmatrix}$ (perturb from $\bar{x}_o$ only in x_2 by $+\Delta x_2$)

$\bar{x}_{2-} = \begin{pmatrix} x_{1,o} \\ x_{2,o} - \Delta x_2 \end{pmatrix}$ (perturb from $\bar{x}_o$ only in x_2 by $-\Delta x_2$)

Δx_2 = scalar perturbation distance (small number)

```
(*First partial*)
Print[" ∂f(x̄o)/∂x1 "]
Print[""]

Δx1 = 0.1;

Print["x1+,x1-"]
x1plus = {{x1o + Δx1}, {x2o}};
x1minus = {{x1o - Δx1}, {x2o}};

x1plus // MatrixForm // N
x1minus // MatrixForm // N
Print[""]

Print["function values"]
fB[x1plus[[1, 1]], x1plus[[2, 1]]] // N
fB[x1minus[[1, 1]], x1minus[[2, 1]]] // N
Print[""]

Print["rise"]
rise = fB[x1plus[[1, 1]], x1plus[[2, 1]]] - fB[x1minus[[1, 1]], x1minus[[2, 1]]]
Print[""]

Print["run"]
run = 2 Δx1
Print[""]

Print["approximate slope"]
rise/run
Print[""]
```

$$\frac{\partial f\left(\bar{x}_o\right)}{\partial x_1}$$

x_{1+}, x_{1-}

$$\begin{pmatrix} 0.6 \\ 0.523599 \end{pmatrix}$$

$$\begin{pmatrix} 0.4 \\ 0.523599 \end{pmatrix}$$

function values

0.126

0.0506667

rise

0.0753333

run

0.2

approximate slope

0.376667

Similarly for the 2nd partial

```
(*Second partial*)
Print["∂f(x̄o)/∂x2 "]
Print[""]

Δx2 = 0.1; (*Note that Δx1 and Δx2 do not need to be the same value*)

Print["x2+,x2- "]
x2plus = {{x1o}, {x2o + Δx2}};
x2minus = {{x1o}, {x2o - Δx2}};

x2plus // MatrixForm // N
x2minus // MatrixForm // N
Print[""]

Print["function values"]
fB[x2plus[[1, 1]], x2plus[[2, 1]]] // N
fB[x2minus[[1, 1]], x2minus[[2, 1]]] // N
Print[""]

Print["rise"]
rise = fB[x2plus[[1, 1]], x2plus[[2, 1]]] - fB[x2minus[[1, 1]], x2minus[[2, 1]]]
Print[""]

Print["run"]
run = 2 Δx2
Print[""]

Print["approximate slope"]
rise/run
Print[""]
∂f(x̄o)/∂x2


x2+,x2-
{{0.5}, {0.623599}}

{{0.5}, {0.423599}}
```

```
function values
```

```
0.0973267
```

```
0.0685073
```

```
rise
```

```
0.0288194
```

```
run
```

```
0.2
```

```
approximate slope
```

```
0.144097
```

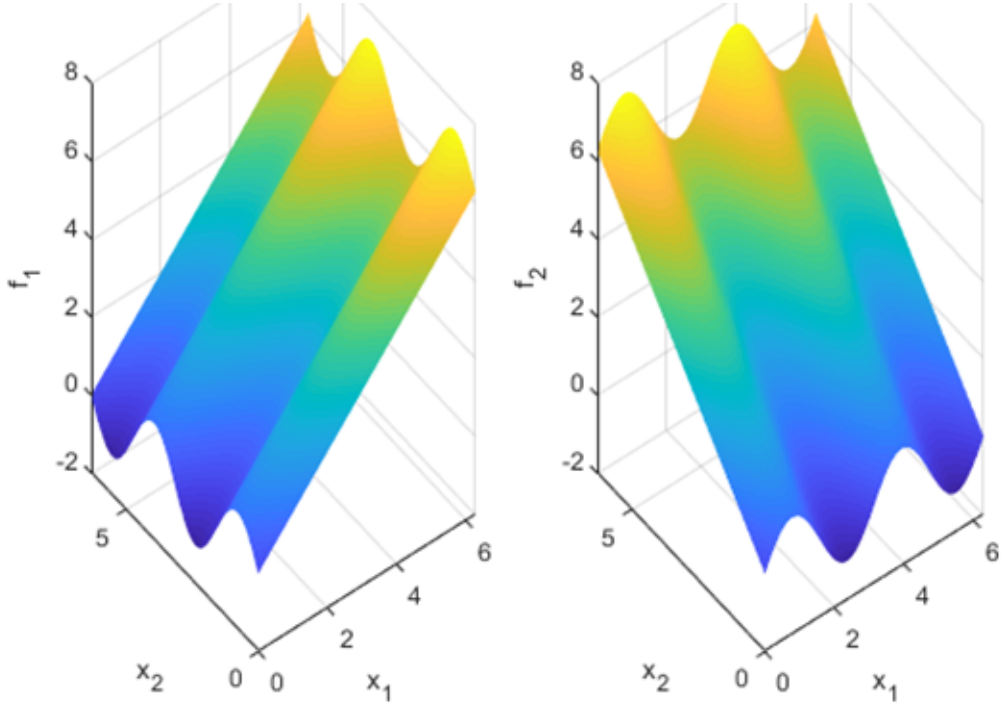
So we obtain

$$\nabla f(\overline{X}_o) = \begin{pmatrix} \frac{\partial f(\overline{X}_o)}{\partial x_1} \\ \frac{\partial f(\overline{X}_o)}{\partial x_2} \end{pmatrix} \approx \begin{pmatrix} 0.376667 \\ 0.144097 \end{pmatrix} \qquad \text{(numerical approximation)}$$

Note that Δx_1 and Δx_2 can be different values. For example, consider the two functions below. As can be seen, f_1 has interesting (sinusoidal) variations in the x_2 direction, but not in the x_1 direction the slope appears constant. Therefore it is possible to choose larger Δx_1 values and still obtain accurate results. However, we need to choose smaller values of Δx_2 in order to obtain the correct slope at the desired point.

The situation is reversed with f_2. We need to keep Δx_1 small but are free to chose larger Δx_2 values.

In summary, the selection of the perturbation distances are dependent on the behavior of the function.



n Variables

We can extend this to n – dimensional functions

$$\nabla f(\overline{x}_o) = \begin{pmatrix} \frac{\partial f(\overline{x}_o)}{\partial x_1} \\ \frac{\partial f(\overline{x}_o)}{\partial x_2} \\ \cdots \\ \frac{\partial f(\overline{x}_o)}{\partial x_n} \end{pmatrix}$$

where each partial is approximated via the symmetric difference quotient as

$$\frac{\partial f(\overline{x}_o)}{\partial x_j} \approx \frac{\text{rise}}{\text{run}} = \frac{f(\overline{x}_{j+}) - f(\overline{x}_{j-})}{2\,\Delta x_j} \qquad \textbf{(Eq.7)}$$

where $\overline{x}_{j+} = \begin{pmatrix} x_{1,o} \\ x_{2,o} \\ \cdots \\ x_{j,o} + \Delta x_j \\ \cdots \\ x_{n,o} \end{pmatrix}$ (perturb from $\overline{x}_o$ only in x_j by $+\Delta x_j$)

$$\overline{x}_{j-} = \begin{pmatrix} x_{1,o} \\ x_{2,o} \\ \cdots \\ x_{j,o} - \Delta x_j \\ \cdots \\ x_{n,o} \end{pmatrix} \qquad \text{(perturb from } \overline{x}_o \text{ only in } x_j \text{ by -}\Delta x_j\text{)}$$

Note that in this lecture, we have written the gradient as a column vector. In some situations, it may be helpful to think of this as a row vector

$$\nabla f(\overline{x}_o) = \begin{pmatrix} \frac{\partial f(\overline{x}_o)}{\partial x_1} & \frac{\partial f(\overline{x}_o)}{\partial x_2} & \cdots & \frac{\partial f(\overline{x}_o)}{\partial x_n} \end{pmatrix}$$

In particular, we will use this row vector formulation when computing Jacobian matrices.