Numerical Differentiation

using the Complex-Step Approximation

Tamas Kis | tamas.a.kis@outlook.com

Tamas Kis
https://tamaskis.github.io

Contents

List of Algorithms

1

The Complex-Step Approximation

1.1 Definition

Consider a scalar-valued function $f : \mathbb{R} \to \mathbb{R}$. The complex-step approximation to its derivative with respect to x, evaluated at $x = x_0$, is defined as [11, 14]

$$\boxed{\left.\frac{df}{dx}\right|_{x=x_0} \approx \frac{\operatorname{Im}\left[f(x_0 + ih)\right]}{h}} \tag{1.1}$$

where h is the step size.

1.2 Implementation

The machine zero, ε, is defined as the smallest positive number ε such that $1 + \varepsilon > 1$ when calculated using a computer). For double precision, $\varepsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ [10, p. 55]. As noted in Cleve Moler's blog post [13] on the topic, the complex-step approximation of a derivative generally converges to within machine zero of the true derivative at a step size of about $h \approx \sqrt{\varepsilon}$ (due to the $\mathcal{O}(h^2)$ convergence). However, [10, p. 234] notes that a step size of 10^{-200} works well for double-precision functions. Therefore, the most basic implementation of the complex-step approximation consists of just two steps: initializing $h = 10^{-200}$, and then approximating the derivative using Eq. (1.1). This procedure is formalized as Algorithm 1 below.

Algorithm 1: `iderivative_s`

Derivative of a univariate, scalar-valued function using the complex-step approximation.

Given:

- $f(x)$ - univariate, scalar-valued function ($f : \mathbb{R} \to \mathbb{R}$)
- $x_0 \in \mathbb{R}$ - point at which to differentiate
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h = 10^{-200}$$

2. Evaluate the derivative of $f(x)$ at x_0 using the complex-step approximation.

$$\left.\frac{df}{dx}\right|_{x=x_0} \approx \frac{\operatorname{Im}\left[f(x_0+ih)\right]}{h}$$

Return:

- $\left.\dfrac{df}{dx}\right|_{x=x_0} \in \mathbb{R}$ - derivative of $f(x)$ evaluated at $x = x_0$

Note: We precede all algorithm names with "`i`" to indicate that the algorithm is using the complex-step derivative approximation, where i is the imaginary unit.

1.3 Complexification and Transposes

Any function that you are trying to differentiate must be able use complex numbers as inputs. Luckily, in MATLAB, most functions can already take complex numbers as inputs. However, there are some special cases of functions where MATLAB will accept complex numbers of inputs, but differentiating them via the complex-step approximation will produce incorrect results. Functions that are known to not be suitable for use with the *Numerical Differentiation Toolbox* (through unit testing; test code is also included with the *Numerical Differentiation Toolbox*) are summarized in Section 1.4.2.

1.3.1 Transposes

It is also vital to note how matrix transposes should be taken when using the complex-step approximation. The transpose operation in MATLAB is typically performed using an apostrophe (`'`), but this is problematic because it actually performs the conjugate transpose (i.e. it also takes the complex conjugate of each element). Therefore, we must use a dot before the apostrophe (`.'`) to perform the non-conjugate transpose [12].

Vector and matrix transposes must be performed using the dot-apostrophe syntax (`.'`).

This also leads to issues with differentiating the standard `norm` and `dot` functions provided by MATLAB. Complexified version of the `norm` and `dot` functions are not included in [2], but we define them in Sections 1.3.3 and 1.3.5.

1.3.2 "Complexified" Functions Included in the Numerical Differentiation Toolbox

The *Numerical Differentiation Toolbox* includes the following functions that have been "complexified" so that they are compatible with the complex-step approximation:

- `iabs` (replaces `abs`)
- `iatan2` (replaces `atan2`)
- `iatan2d` (replaces `atan2d`)
- `idot` (replaces `dot`)

- `imax` (replaces `max` for the maximum of two numbers)
- `imin` (replaces `min` for the minimum of two numbers)
- `imod` (replaces `mod` for a complex dividend and real divisor)
- `inorm` (replaces `norm` for 2-norm of a vector)

The "complexifications" of `abs`, `atan2`, `atan2d`, `max`, and `min` are adapted from a Fortran implementation [2]. [2] also includes additional complexified functions that have not been included here. The remaining complexifications (of `dot`, `mod`, and `norm`) are discussed in Sections 1.3.3–1.3.5.

1.3.3 `idot`

For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the dot product and 2-norm are defined as

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y}$$

However, MATLAB's `dot` function assumes $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ and uses

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^H \mathbf{y}$$

In the *Numerical Differentiation Toolbox*, `idot` (the "complexified" version of `dot`) is implemented using $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y}$.

1.3.4 `imod`

The modulo operator, mod, returns the remainder, r, of a/n. MATLAB implements this as

$$\mathrm{mod}\,(a, n) = a - \left\lfloor \frac{a}{n} \right\rfloor n$$

where $\lfloor \cdot \rfloor$ represents the floor functions. While MATLAB's `floor` option accepts complex-valued inputs, its `mod` function does not. Therefore, we simply define the `imod` function using the definition above.

Unit testing of the `imod` function is not included in the *Numerical Differentiation Toolbox*. However, it was tested against a finite-difference approximation when differentiating a (rather complicated) function that used the modulo operation with $n \in \mathbb{R}$. Therefore, for the *Numerical Differentiation Toolbox* implementation of `imod`, it should be noted that n should be real-valued.

1.3.5 `inorm`

For $\mathbf{x} \in \mathbb{R}^n$, 2-norm is defined as

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}}$$

However, MATLAB's `dot` function assumes $\mathbf{x} \in \mathbb{C}^n$ and uses

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^H \mathbf{x}}$$

In the *Numerical Differentiation Toolbox*, `inorm` (the "complexified" version of `norm` specifically for the 2-norm of vectors) is implemented using $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}}$.

1.4 Limitations

First and foremost, the complex-step approximation is only intended for approximating the derivatives of ***real***-valued functions (as evident by its definition in Section 1.1. However, there are additional limitations discussed in Sections 1.4.1 and 1.4.2 that should be noted.

1.4.1 Higher-Order Derivatives

A proposed extension of the complex-step approximation to second derivatives is

$$\left.\frac{df}{dx}\right|_{x=x_0} \approx \frac{2\left(f(x_0) - \mathrm{Re}\left[f(x_0 + ih)\right]\right)}{h^2}$$

Unfortunately, unlike in the first derivative case, the second derivative approximation can introduce errors if the step size is made too small [9]. Therefore, when evaluating second derivatives while forming the Hessian matrix in Section 2.6, we will use a different approach that utilizes a hybrid of complex-step and central difference approximations.

Additionally, we cannot use nested calls on a complex-step differentiation algorithm to obtain higher-order derivatives. Consider trying to approximate a second derivative by nesting one complex-step approximation within another:

$$\left.\frac{df}{dx}\right|_{x=x_0} \approx \left.\frac{d}{dx}\right|_{x=x_0} \left[\frac{\mathrm{Im}\left[f(x_0 + ih)\right]}{h}\right] \approx \frac{\mathrm{Im}\overbrace{\left[\frac{\mathrm{Im}\left[f(x_0 + 2ih)\right]}{h}\right]}^{\text{this term has no imaginary part}}}{h}$$

Since the term in the bracket has no imaginary part, we would simply get

$$\left.\frac{df}{dx}\right|_{x=x_0} = 0$$

which is incorrect.

1.4.2 Functions That Yield Incorrect Results

The following scalar-valued functions were tested in `iderivative_test` in the *Numerical Differentiation Toolbox* for MATLAB and yield incorrect results:

- $\mathrm{arccsc}\,(x)$ for $x < -1$
- $\mathrm{arcsec}\,(x)$ for $x < -1$
- $\mathrm{arccoth}\,(x)$ for $0 < x < 1$
- $\mathrm{arctanh}\,(x)$ for $x > 1$
- $\mathrm{arcsech}\,(x)$ for $-1 < x < 0$
- $\mathrm{arccoth}\,(x)$ for $-1 < x < 0$
- $\mathrm{arccosh}\,(x)$ for $x < -1$
- $\mathrm{arctanh}\,(x)$ for $x < -1$

> **WARNING:** These errors remain uncorrected in the *Numerical Differentiation Toolbox.*

"Complexified" versions of most functions can be found in [2] (as discussed in Section 1.3.2).

2

Differentiation Using the Complex-Step Approximation

2.1 Derivative of a Univariate, Vector-Valued Function

Consider a univariate, vector-valued function $\mathbf{f}:\mathbb{R}\to\mathbb{R}^m$. To approximate its derivative at a point $x=x_0$, we can begin by simply differentiating each component of $\mathbf{f}$ with respect to x and evaluating the result at $x=x_0$.

$$\left.\frac{d\mathbf{f}}{dx}\right|_{x=x_0}=\begin{bmatrix}\left.\dfrac{df_1}{dx}\right|_{x=x_0}\\ \vdots\\ \left.\dfrac{df_m}{dx}\right|_{x=x_0}\end{bmatrix}$$

Applying the complex-step approximation to evaluate at $x=x_0$,

$$\left.\frac{d\mathbf{f}}{dx}\right|_{x=x_0}\approx\begin{bmatrix}\dfrac{\operatorname{Im}\left[f_1(x_0+ih)\right]}{h}\\ \vdots\\ \dfrac{\operatorname{Im}\left[f_m(x_0+ih)\right]}{h}\end{bmatrix}$$

Therefore, in a more compact form, we can simply write

$$\boxed{\left.\frac{d\mathbf{f}}{dx}\right|_{x=x_0}\approx\frac{\operatorname{Im}\left[\mathbf{f}(x_0+ih)\right]}{h}}\tag{2.1}$$

For completeness, we include Algorithm 2. However, it should be noted that Algorithm 2 is nearly identical to Algorithm 1, except it is generalized to a vector-valued function $\mathbf{f}$.

Algorithm 2: `iderivative`

Derivative of a univariate, vector-valued function using the complex-step approximation.

Given:

- $\mathbf{f}(x)$ - univariate, vector-valued function ($\mathbf{f}:\mathbb{R}\to\mathbb{R}^m$)
- $x_0\in\mathbb{R}$ - point at which to differentiate
- $h\in\mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h=10^{-200}$$

2. Evaluate the derivative of $\mathbf{f}(x)$ at $x=x_0$ using the complex-step approximation.

$$\left.\frac{d\mathbf{f}}{dx}\right|_{x=x_0}\approx\frac{\operatorname{Im}\left[\mathbf{f}(x_0+ih)\right]}{h}$$

Return:

- $\left.\dfrac{d\mathbf{f}}{dx}\right|_{x=x_0}\in\mathbb{R}^m$ - derivative of $\mathbf{f}(x)$ evaluated at x_0

Note:

- This function requires 1 evaluation of $\mathbf{f}(x)$.

2.2 Partial Derivative of a Multivariate, Scalar or Vector-Valued Function

Consider a multivariate, scalar-valued function $f:\mathbb{R}^n\to\mathbb{R}$ where $\mathbf{x}\in\mathbb{R}^n$ is the independent variable. The complex-step approximation for the partial derivative of $f(\mathbf{x})$ with respect to x_j, evaluated at $\mathbf{x}=\mathbf{x}_0$, is

$$\left.\frac{\partial f}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}\approx\frac{\operatorname{Im}\left[f\left((x_{0,1},x_{0,2},...,x_{0,j}+ih,...,x_{0,n})^T\right)\right]}{h}\tag{2.2}$$

If we define

$$\boxed{\boldsymbol{\chi}_j=\begin{bmatrix}0\\ \vdots\\ ih\\ \vdots\\ 0\end{bmatrix}\leftarrow j^{\text{th}}\text{ element}}\tag{2.3}$$

then we can write the partial derivative in a more compact form as

$$\boxed{\left.\frac{\partial f}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}\approx\frac{\operatorname{Im}\left[f(\mathbf{x}_0+\boldsymbol{\chi}_j)\right]}{h}}\tag{2.4}$$

From Section 2.1, we know it is trivial to extend this to the vector-valued case ($\mathbf{f}:\mathbb{R}^n\to\mathbb{R}^m$):

$$\boxed{\left.\frac{\partial \mathbf{f}}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}\approx\frac{\operatorname{Im}\left[\mathbf{f}(\mathbf{x}_0+\boldsymbol{\chi}_j)\right]}{h}}\tag{2.5}$$

In its implementation, we do not use $\boldsymbol{\chi}_j$ (although this term will be useful in later sections). Instead, within the function, we redefine $\mathbf{x}_0$ as

$$\mathbf{x}_0 \leftarrow \mathbf{x}_0 + \boldsymbol{\chi}_j$$

and then simply perform

$$\left.\frac{\partial \mathbf{f}}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \approx \frac{\text{Im}\left[\mathbf{f}(\mathbf{x}_0)\right]}{h}$$

Algorithm 3: `ipartial`

Partial derivative of a multivariate, scalar or vector-valued function using the complex-step approximation.

Given:

- $\mathbf{f}(\mathbf{x})$ - multivariate, vector-valued function ($\mathbf{f}:\mathbb{R}^n \to \mathbb{R}^m$)
- $\mathbf{x}_0 \in \mathbb{R}^n$ - point at which to differentiate
- $j \in \mathbb{Z}$ - index of element of $\mathbf{x}$ to differentiate with respect to
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h = 10^{-200}$$

2. Redefine $\mathbf{x}_0$ by updating its j^{th} element.

$$x_{0,j} = x_{0,j} + ih$$

3. Evaluate the partial derivative of $\mathbf{f}(\mathbf{x})$ with respect to x_j at $\mathbf{x} = \mathbf{x}_0$ using the complex-step approximation.

$$\left.\frac{\partial \mathbf{f}}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \approx \frac{\text{Im}\left[\mathbf{f}(\mathbf{x}_0)\right]}{h}$$

Return:

- $\left.\frac{\partial \mathbf{f}}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \in \mathbb{R}^m$ - partial derivative of $\mathbf{f}(\mathbf{x})$ with respect to x_j, evaluated at $\mathbf{x} = \mathbf{x}_0$

Note:

- This algorithm can be used for a scalar-valued function by just inputting a function where $m = 1$ (i.e. $f:\mathbb{R}^n \to \mathbb{R}$ instead of $\mathbf{f}:\mathbb{R}^n \to \mathbb{R}^m$).
- This function requires 1 evaluation of $\mathbf{f}(\mathbf{x})$.

2.3 Gradient

Consider a multivariate, scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$. The gradient of f with respect to $\mathbf{x} \in \mathbb{R}^n$, evaluated at $\mathbf{x} = \mathbf{x}_0$, is defined as [4]

$$\boxed{\mathbf{g}(\mathbf{x}_0) = \nabla f(\mathbf{x}_0) = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} \left.\frac{\partial f}{\partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} \\ \vdots \\ \left.\frac{\partial f}{\partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \end{bmatrix}} \tag{2.6}$$

To evaluate the gradient, we need to approximate all of the partial derivatives. From Eq. (2.4), we know the j^{th} partial derivative can be approximated as

$$\frac{\partial f}{\partial x_j} \approx \frac{\operatorname{Im}\left[f(\mathbf{x} + \boldsymbol{\chi}_j)\right]}{h}$$

where

$$\boldsymbol{\chi}_j = \begin{bmatrix} 0 \\ \vdots \\ ih \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow j^{\text{th}} \text{ element} \\ \\ \\ \end{matrix}$$

To make the computational implementation a bit neater and more efficient, we introduce the complex-step matrix, $\mathbf{X}$, defined as

$$\boxed{\mathbf{X} = ih\mathbf{I}_{n\times n} = \begin{bmatrix} ih & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & ih \end{bmatrix} = \begin{bmatrix} | & & | \\ \boldsymbol{\chi}_1 & \dots & \boldsymbol{\chi}_n \\ | & & | \end{bmatrix}} \tag{2.7}$$

As evident from Eq. (2.7), $\boldsymbol{\chi}_j$ is simply the j^{th} column of $\mathbf{X}$. Now, we can develop the algorithm to approximate the gradient.

Algorithm 4: `igradient`

Gradient of a multivariate, scalar-valued function using the complex-step approximation.

Given:

- $f(\mathbf{x})$ - multivariate, scalar-valued function ($f : \mathbb{R}^n \to \mathbb{R}$)
- $\mathbf{x}_0 \in \mathbb{R}^n$ - point at which to evaluate the gradient
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h = 10^{-200}$$

2. Determine n from the fact that $\mathbf{x}_0 \in \mathbb{R}^n$.
3. Preallocate a vector $\mathbf{g} \in \mathbb{R}^n$ to store the gradient.
4. Define the complex-step matrix.

$$\mathbf{X} = ih\mathbf{I}_{n\times n}$$

5. Evaluate the gradient.

for $j = 1$ **to** n

$$g_j = \frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_j)\right]}{h} \quad \text{(where } \boldsymbol{\chi}_j \text{ is the } j^{\text{th}} \text{ column of } \mathbf{X}\text{)}$$

end

Return:

- $\mathbf{g} \in \mathbb{R}^n$ - gradient of f evaluated at $\mathbf{x} = \mathbf{x}_0$

Note:

- This function requires n evaluations of $f(\mathbf{x})$.

2.4 Directional Derivative

Consider a multivariate, scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$. The directional derivative of f at $\mathbf{x} = \mathbf{x}_0$ in the direction of $\mathbf{v} \in \mathbb{R}^n$ is [3]

$$\boxed{D_{\mathbf{v}} f(\mathbf{x}_0) = \mathbf{v} \cdot \nabla f(\mathbf{x}_0)} \tag{2.8}$$

Therefore, we can simply add a single additional line to Algorithm 4 to develop Algorithm 5 to calculate the directional derivative.

Algorithm 5: `idirectional`
Directional derivative of a multivariate, scalar-valued function using the complex-step approximation.

Given:

- $f(\mathbf{x})$ - multivariate, scalar-valued function ($f : \mathbb{R}^n \to \mathbb{R}$)
- $\mathbf{x}_0 \in \mathbb{R}^n$ - point at which to evaluate the directional derivative
- $\mathbf{v} \in \mathbb{R}^n$ - vector defining direction of differentiation
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h = 10^{-200}$$

2. Determine n from the fact that $\mathbf{x}_0 \in \mathbb{R}^n$.
3. Preallocate a vector $\mathbf{g} \in \mathbb{R}^n$ to store the gradient.
4. Define the complex-step matrix.

$$\mathbf{X} = ih\mathbf{I}_{n \times n}$$

5. Evaluate the gradient.

for $j = 1$ **to** n

$$g_j = \frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_j)\right]}{h} \quad \text{(where } \boldsymbol{\chi}_j \text{ is the } j^{\text{th}} \text{ column of } \mathbf{X}\text{)}$$

end

6. Evaluate the directional derivative.

$$D_{\mathbf{v}} = \mathbf{v} \cdot \mathbf{g}$$

Return:
- $D_{\mathbf{v}} \in \mathbb{R}$ - directional derivative of f evaluated at $\mathbf{x} = \mathbf{x}_0$ in the direction of $\mathbf{v}$

Note:
- This function requires n evaluations of $f(\mathbf{x})$.

2.5 Jacobian Matrix

The Jacobian matrix of a multivariate, vector-valued function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ with respect to $\mathbf{x} \in \mathbb{R}^n$, evaluated at $\mathbf{x} = \mathbf{x}_0$, is defined as

$$\boxed{\mathbf{J}(\mathbf{x}_0) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} \left.\frac{\partial f_1}{\partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\frac{\partial f_1}{\partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial f_m}{\partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\frac{\partial f_m}{\partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \end{bmatrix}} \tag{2.9}$$

It is helpful to rewrite Eq. (2.9) in terms of its column vectors as [8]

$$\boxed{\mathbf{J}(\mathbf{x}_0) = \begin{bmatrix} \left.\frac{\partial \mathbf{f}}{\partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\frac{\partial \mathbf{f}}{\partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \end{bmatrix}} \tag{2.10}$$

From Eq. (2.5), we know

$$\left.\frac{\partial \mathbf{f}}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \approx \frac{\operatorname{Im}\left[\mathbf{f}(\mathbf{x}_0 + \boldsymbol{\chi}_j)\right]}{h}$$

where

$$\boldsymbol{\chi}_j = \begin{bmatrix} 0 \\ \vdots \\ ih \\ \vdots \\ 0 \end{bmatrix} \leftarrow j^{\text{th}} \text{ element}$$

Utilizing Eq. (2.10), we only need to make a small adjustment to Algorithm 4 to develop Algorithm 6 for evaluating the Jacobian using the complex-step approximation.

Algorithm 6: `ijacobian`

Jacobian matrix of a multivariate, vector-valued function using the complex-step approximation.

Given:
- $\mathbf{f}(\mathbf{x})$ - multivariate, vector-valued function ($\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$)
- $\mathbf{x}_0 \in \mathbb{R}^n$ - point at which to evaluate the Jacobian matrix
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

$$h = 10^{-200}$$

2. Determine m from the fact that $\mathbf{f}(\mathbf{x}_0) \in \mathbb{R}^m$.
3. Determine n from the fact that $\mathbf{x}_0 \in \mathbb{R}^n$.
4. Preallocate a matrix $\mathbf{J} \in \mathbb{R}^{m \times n}$ to store the Jacobian.
5. Define the complex-step matrix.

$$\mathbf{X} = ih\mathbf{I}_{n \times n}$$

6. Evaluate the Jacobian matrix (where $\mathbf{j}_j$ is the j^{th} column vector of $\mathbf{J}$).

for $j = 1$ **to** n

$$\mathbf{j}_j = \frac{\text{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_j)\right]}{h} \quad \text{(where } \boldsymbol{\chi}_j \text{ is the } j^{\text{th}} \text{ column of } \mathbf{X}\text{)}$$

end

Return:

- $\mathbf{J} \in \mathbb{R}^{m \times n}$ - Jacobian matrix of $\mathbf{f}$ evaluated at $\mathbf{x} = \mathbf{x}_0$

Note:

- This function requires $n + 1$ evaluations of $\mathbf{f}(\mathbf{x})$.

2.6 Hessian Matrix

The Hessian matrix of a multivariate, scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to $\mathbf{x} \in \mathbb{R}^n$, evaluated at $\mathbf{x} = \mathbf{x}_0$, is defined as

$$\boxed{\mathbf{H}(\mathbf{x}_0) = \begin{bmatrix} \left.\dfrac{\partial^2 f}{\partial x_1^2}\right|_{\mathbf{x}=\mathbf{x}_0} & \left.\dfrac{\partial^2 f}{\partial x_1 \partial x_2}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\dfrac{\partial^2 f}{\partial x_1 \partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \\ \left.\dfrac{\partial^2 f}{\partial x_2 \partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} & \left.\dfrac{\partial^2 f}{\partial x_2^2}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\dfrac{\partial^2 f}{\partial x_2 \partial x_n}\right|_{\mathbf{x}=\mathbf{x}_0} \\ \vdots & \vdots & \ddots & \vdots \\ \left.\dfrac{\partial^2 f}{\partial x_n \partial x_1}\right|_{\mathbf{x}=\mathbf{x}_0} & \left.\dfrac{\partial^2 f}{\partial x_n \partial x_2}\right|_{\mathbf{x}=\mathbf{x}_0} & \cdots & \left.\dfrac{\partial^2 f}{\partial x_n^2}\right|_{\mathbf{x}=\mathbf{x}_0} \end{bmatrix}} \tag{2.11}$$

In a more compact form, we can write

$$\boxed{\left[\mathbf{H}(\mathbf{x}_0)\right]_{j,k} = \frac{\partial^2 f}{\partial x_j \partial x_k}} \tag{2.12}$$

From Schwarz's theorem, we know

$$\frac{\partial^2 f}{\partial x_j \partial x_k} = \frac{\partial^2 f}{\partial x_k \partial x_j}$$

which implies that the Hessian matrix is symmetric and satisfies the property [6]

$$\boxed{\left[\mathbf{H}(\mathbf{x}_0)\right]_{j,k} = \left[\mathbf{H}(\mathbf{x}_0)\right]_{k,j}} \tag{2.13}$$

As mentioned in Section 1.4.1, we will not be using a true complex-step approximation for the second derivative in this section. Instead, we will evaluate the first derivative using a complex-step approximation, and the second derivative using a central difference approximation. Beginning by rewriting Eq. (2.12) in a slightly different form,

$$[\mathbf{H}(\mathbf{x}_0)]_{j,k} = \left.\frac{\partial}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \left(\left.\frac{\partial f}{\partial x_k}\right|_{\mathbf{x}=\mathbf{x}_0} \right)$$

Replacing the derivative in the parentheses with its complex-step approximation (from Eq. (2.4)), we can write

$$[\mathbf{H}(\mathbf{x}_0)]_{j,k} \approx \left.\frac{\partial}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \left(\frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k)\right]}{h} \right) \tag{2.14}$$

where

$$\boldsymbol{\chi}_k = \begin{bmatrix} 0 \\ \vdots \\ ih \\ \vdots \\ 0 \end{bmatrix} \leftarrow k^{\text{th}} \text{ element}$$

The central difference approximation to a derivative is [15]

$$\left.\frac{df}{dx}\right|_{x=x_0} \approx \frac{f(x_0+h) - f(x_0-h)}{2h}$$

For a multivariate function, this can be written as

$$\left.\frac{\partial f}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0} \approx \frac{f(\mathbf{x}_0 + \mathbf{u}_j) - f(\mathbf{x}_0 - \mathbf{u}_j)}{2h} \tag{2.15}$$

where

$$\boxed{\mathbf{u}_j = \begin{bmatrix} 0 \\ \vdots \\ h \\ \vdots \\ 0 \end{bmatrix} \leftarrow j^{\text{th}} \text{ element}} \tag{2.16}$$

Applying the approximation from Eq. (2.15) to Eq. (2.14),

$$\begin{aligned}
[\mathbf{H}(\mathbf{x}_0)]_{j,k} &\approx \frac{1}{2h}\left[\left(\frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k) + \mathbf{u}_j\right]}{h} \right) - \left(\frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k - \mathbf{u}_j)\right]}{h} \right) \right] \\
&\approx \frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k + \mathbf{u}_j)\right] - \mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k - \mathbf{u}_j)\right]}{2h^2} \\
&\approx \frac{\mathrm{Im}\left[f(\mathbf{x}_0 + \boldsymbol{\chi}_k + \mathbf{u}_j) - f(\mathbf{x}_0 + \boldsymbol{\chi}_k - \mathbf{u}_j)\right]}{2h^2}
\end{aligned}$$

Finally, let's define

$$\boxed{\mathbf{x}_{r,k} = \mathbf{x}_0 + \boldsymbol{\chi}_k} \tag{2.17}$$

$$\boxed{\mathbf{y}_{j,k}^{+} = \mathbf{x}_{r,k} + \mathbf{u}_j} \tag{2.18}$$

$$\boxed{\mathbf{y}_{j,k}^{-} = \mathbf{x}_{r,k} - \mathbf{u}_j} \tag{2.19}$$

Then we arrive at the result

$$\boxed{[\mathbf{H}(\mathbf{x}_0)]_{j,k} = [\mathbf{H}(\mathbf{x}_0)]_{k,j} \approx \frac{\mathrm{Im}\left[f\left(\mathbf{y}^+_{j,k}\right) - f\left(\mathbf{y}^-_{j,k}\right)\right]}{2h^2}} \tag{2.20}$$

Similar to the complex-step matrix $\mathbf{X}$, we define the real-step matrix $\mathbf{U}$ as

$$\boxed{\mathbf{U} = h\mathbf{I}_{n\times n} = \begin{bmatrix} h & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & h \end{bmatrix} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_n \\ | & & | \end{bmatrix}} \tag{2.21}$$

With these definitions, we can develop Algorithm 7 below[1]. Since the Hessian matrix is symmetric, we only have to evaluate the derivatives in the upper triangle of the matrix (shown in red below):

$$\begin{bmatrix} H_{1,1} & H_{1,2} & \dots & H_{1,n} \\ H_{2,1} & H_{2,2} & \dots & H_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n,1} & H_{n,2} & \dots & H_{n,n} \end{bmatrix}$$

Algorithm 7 uses a default step size of $h = \varepsilon^{1/3}$, where ε represents machine zero (see Section 1.2). While the complex-step approximation can use $h \ll \sqrt{\varepsilon}$, the optimal step size for the central difference approximation scales approximately with $\varepsilon^{1/3}$ [10, p. 230].

Algorithm 7: `ihessian`

Hessian matrix of a multivariate, scalar-valued function using the complex-step and central difference approximations.

Given:

- $f(\mathbf{x})$ - multivariate, scalar-valued function ($f : \mathbb{R}^n \to \mathbb{R}$)
- $\mathbf{x}_0 \in \mathbb{R}^n$ - point at which to evaluate the Hessian matrix
- $h \in \mathbb{R}$ - *(OPTIONAL)* step size

Procedure:

1. Initialize the step size if not input.

   $$h = \varepsilon^{1/3}$$

2. Determine n from the fact that $\mathbf{x}_0 \in \mathbb{R}^n$.
3. Preallocate a matrix $\mathbf{H} \in \mathbb{R}^{n\times n}$ to store the Hessian.
4. Define the complex-step and real-step matrices.

   $$\mathbf{X} = ih\mathbf{I}_{n\times n}$$
   $$\mathbf{U} = h\mathbf{I}_{n\times n}$$

5. Loop over each independent variable.

   for $k = 1$ **to** n

1 This algorithm was inspired by [1] and [5].

(a) Define the reference point with the complex increment in the k^{th} independent variable.

$$\mathbf{x}_r = \mathbf{x}_0 + \boldsymbol{\chi}_k$$

(b) Loop through the upper triangular elements.

for $j = k$ **to** n

i. Define the reference points with the real increments in the j^{th} independent variable.

$$\mathbf{y}^+ = \mathbf{x}_r + \mathbf{u}_j$$
$$\mathbf{y}^- = \mathbf{x}_r - \mathbf{u}_j$$

ii. Evaluate the $(k, j)^{\text{th}}$ element of the Hessian.

$$H_{k,j} \approx \frac{\text{Im}\left[f(\mathbf{y}^+) - f(\mathbf{y}^-)\right]}{2h^2}$$

iii. Evaluate the $(j, k)^{\text{th}}$ element of the Hessian using symmetry.

$$H_{j,k} = H_{k,j}$$

end

end

Return:

- $\mathbf{H} \in \mathbb{R}^{n \times n}$ - Hessian matrix of f evaluated at $\mathbf{x} = \mathbf{x}_0$

Note:

- $\boldsymbol{\chi}_k$ is the k^{th} column of $\mathbf{X}$ and $\mathbf{u}_j$ is the j^{th} column of $\mathbf{U}$.
- This function requires $n(n+1)$ evaluations of $f(\mathbf{x})$ (the upper triangular matrix entries (including the diagonal) consist of $n(n+1)/2$ entries [7], and each entry requires 2 evaluations of $f(\mathbf{x})$).

References

[1] Yi Cao. *Complex step Hessian*. MATLAB Central File Exchange. Accessed: August 3, 2021. URL: `https://www.mathworks.com/matlabcentral/fileexchange/18177-complex-step-hessian?s_tid=srchtitle`.

[2] *complexify.f90*. MDO Lab. Accessed: August 3, 2021. URL: `https://mdolab.engin.umich.edu/misc/files/complexify.f90`.

[3] *Directional derivative*. Wikipedia. Accessed: August 3, 2021. URL: `https://en.wikipedia.org/wiki/Directional_derivative`.

[4] *Gradient*. Wikipedia. Accessed: August 3, 2021. URL: `https://en.wikipedia.org/wiki/Gradient`.

[5] Daniel R. Herberg. `DTQP_hessian_complex_step` *from DT QP Project*. MATLAB Central File Exchange. Accessed: August 3, 2021. URL: `https://www.mathworks.com/matlabcentral/fileexchange/65434-dt-qp-project?s_tid=srchtitle`.

[6] *Hessian matrix*. Wikipedia. Accessed: August 3, 2021. URL: `https://en.wikipedia.org/wiki/Hessian_matrix`.

[7] *Is there an analytic expression for a number of elements inside a triangular matrix (with and without items on diagonal)*. Stack Exchange. Accessed: December 26, 2021. URL: `https://math.stackexchange.com/questions/2388887/is-there-an-analytic-expression-for-a-number-of-elements-inside-a-triangular-mat/2388889`.

[8] *Jacobian matrix and determinant*. Wikipedia. Accessed: August 3, 2021. URL: `https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant`.

[9] Kok-Lam Lai and John L. Crassidis. “Generalizations of the Complex-Step Derivative Approximation”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit* (2006). DOI: `10.2514/6.2006-6348`.

[10] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, 2021. URL: `https://mdobook.github.io/`.

[11] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. “The Complex-Step Derivative Approximation”. In: *ACM Transactions on Mathematical Software* 29.3 (Sept. 2003), pp. 245–262. DOI: `10.1145/838250.838251`.

[12] *Matlab Implementation*. MDO Lab. Accessed: August 3, 2021. URL: `https://mdolab.engin.umich.edu/misc/complex-step-guide-matlab`.

[13] Cleve Moler. *Complex Step Differentiation*. MathWorks. Accessed: August 3, 2021. URL: `https://blogs.mathworks.com/cleve/2013/10/14/complex-step-differentiation/`.

[14] William Squire and George Trapp. “Using Complex Variables to Estimate Derivatives of Real Functions”. In: *SIAM Review* 40.1 (Mar. 1998), pp. 110–112. DOI: `10.1137/S003614459631241X`.

[15] Todd Young and Martin J. Mohlenkamp. *Lecture 27: Numerical Differentiation*. Introduction to Numerical Methods and Matlab Programming for Engineers. Accessed: August 3, 2021. URL: `http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture27.pdf`.