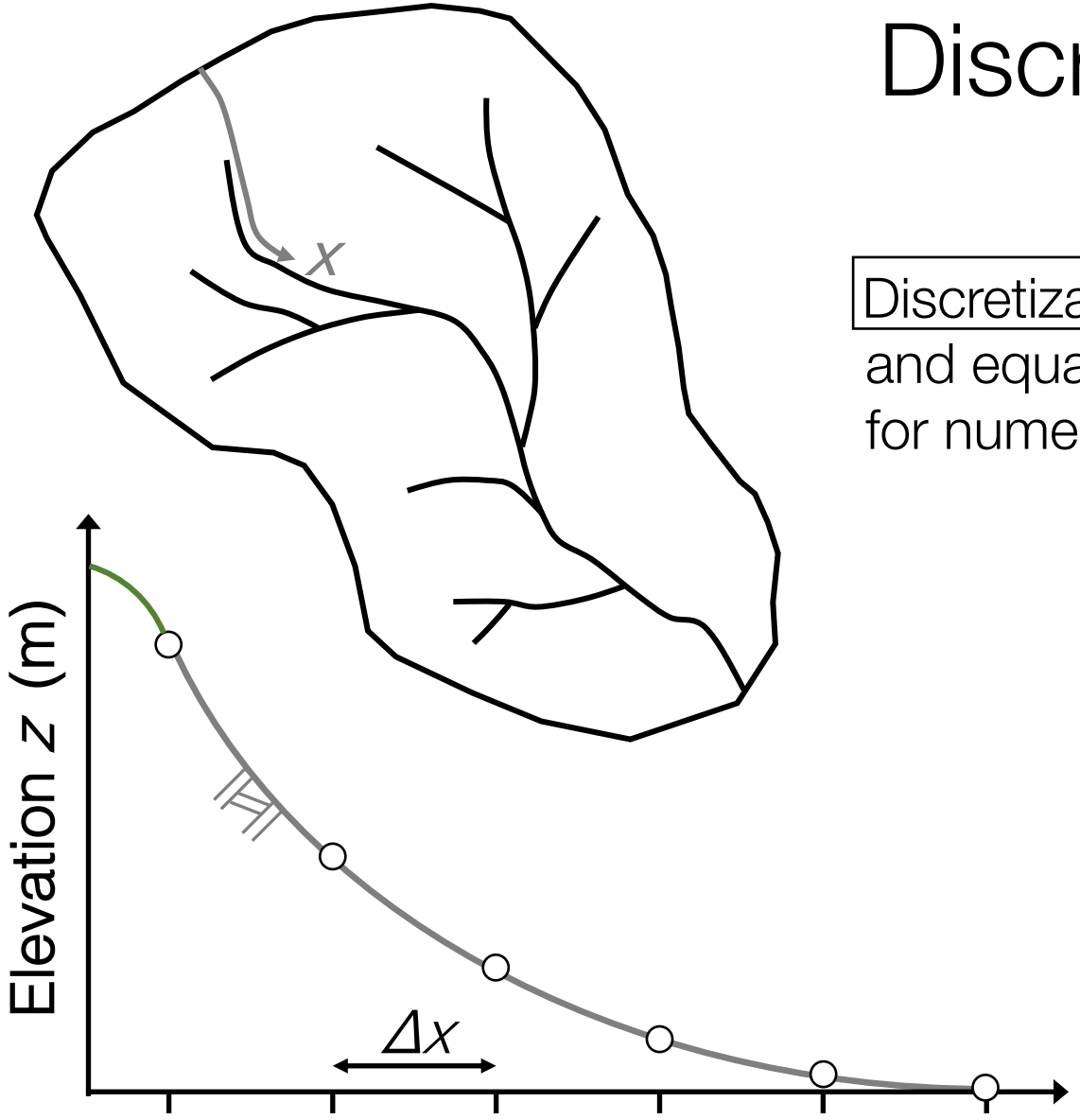
ESPM: 1D river profile evolution

1. Brief review of differential equations
2. Bedrock river incision
3. Discretizing a river profile
   Exercise 1: Discretizing a river profile
4. Hack's law
   Exercise 2: Drainage area along the river profile
5. Stream power law and the steady-state SPL river profile
   Exercise 3: Steady-state river profile
6. Finite differencing in space
   Exercise 4: Finite difference approximation of the spatial derivative

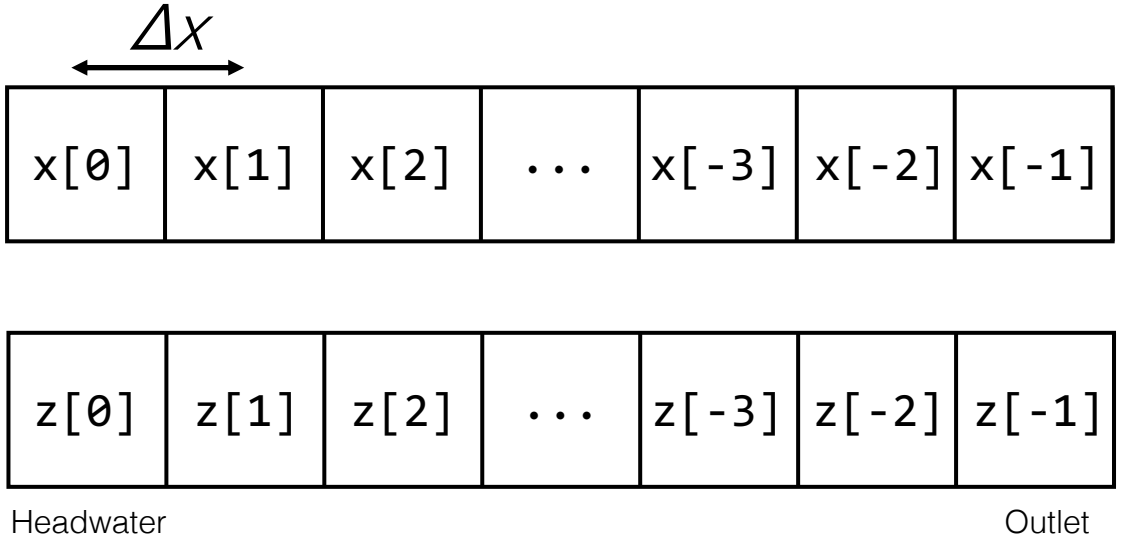
Week 1

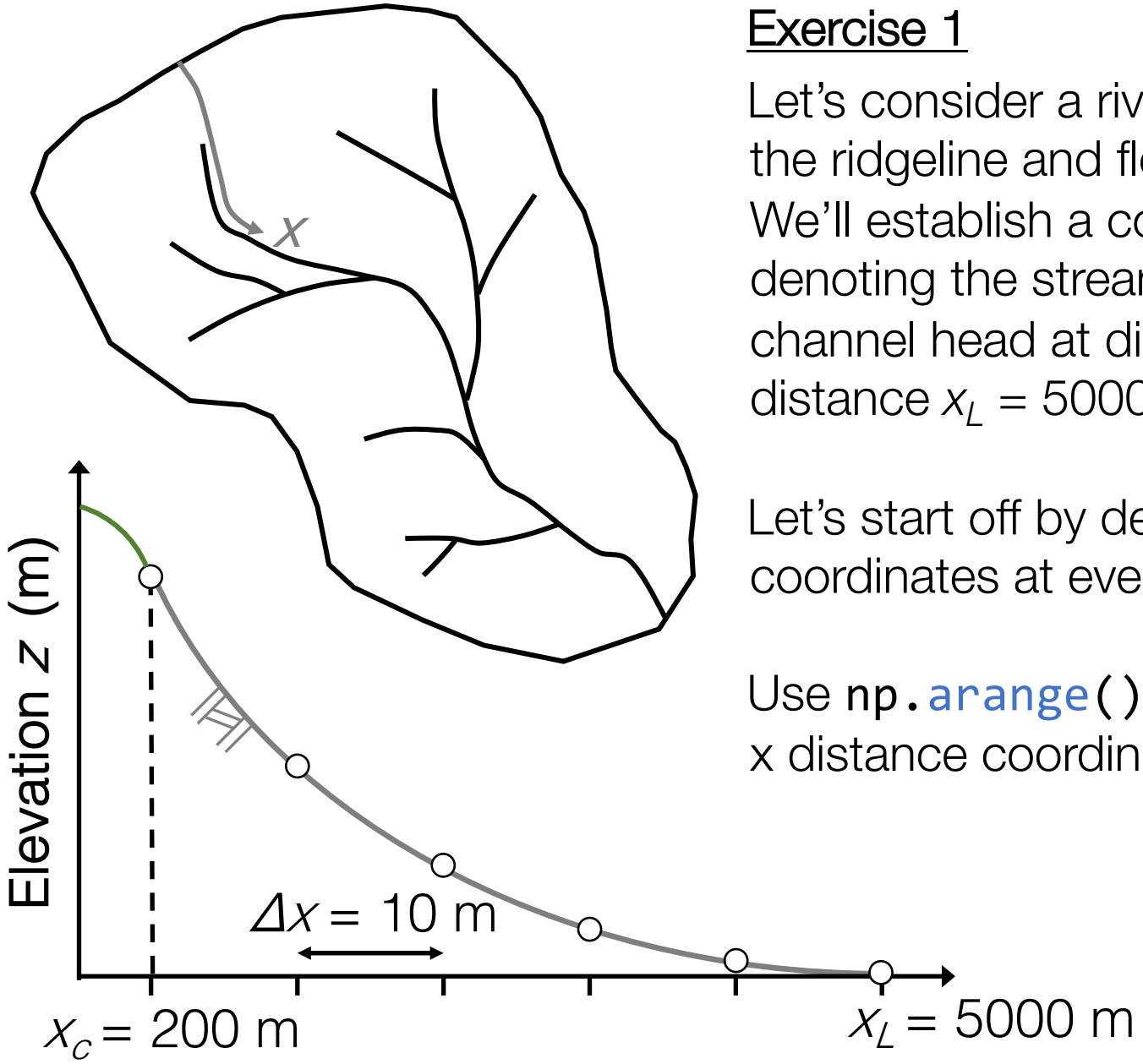
7. Finite differencing in time
   Exercise 5: River profile update
8. River profile evolution
   Exercise 6: River profile evolution
9. Accuracy, stability, and the CFL condition
   Exercise 7: CFL condition for the SPL evolution equation
10. Model evaluation
    Exercise 8: Numerical vs. analytic steady-state river profiles

Discretization



Discretization is the representation of continuous variables and equations by individual elements – making them suitable for numerical computation





Exercise 1

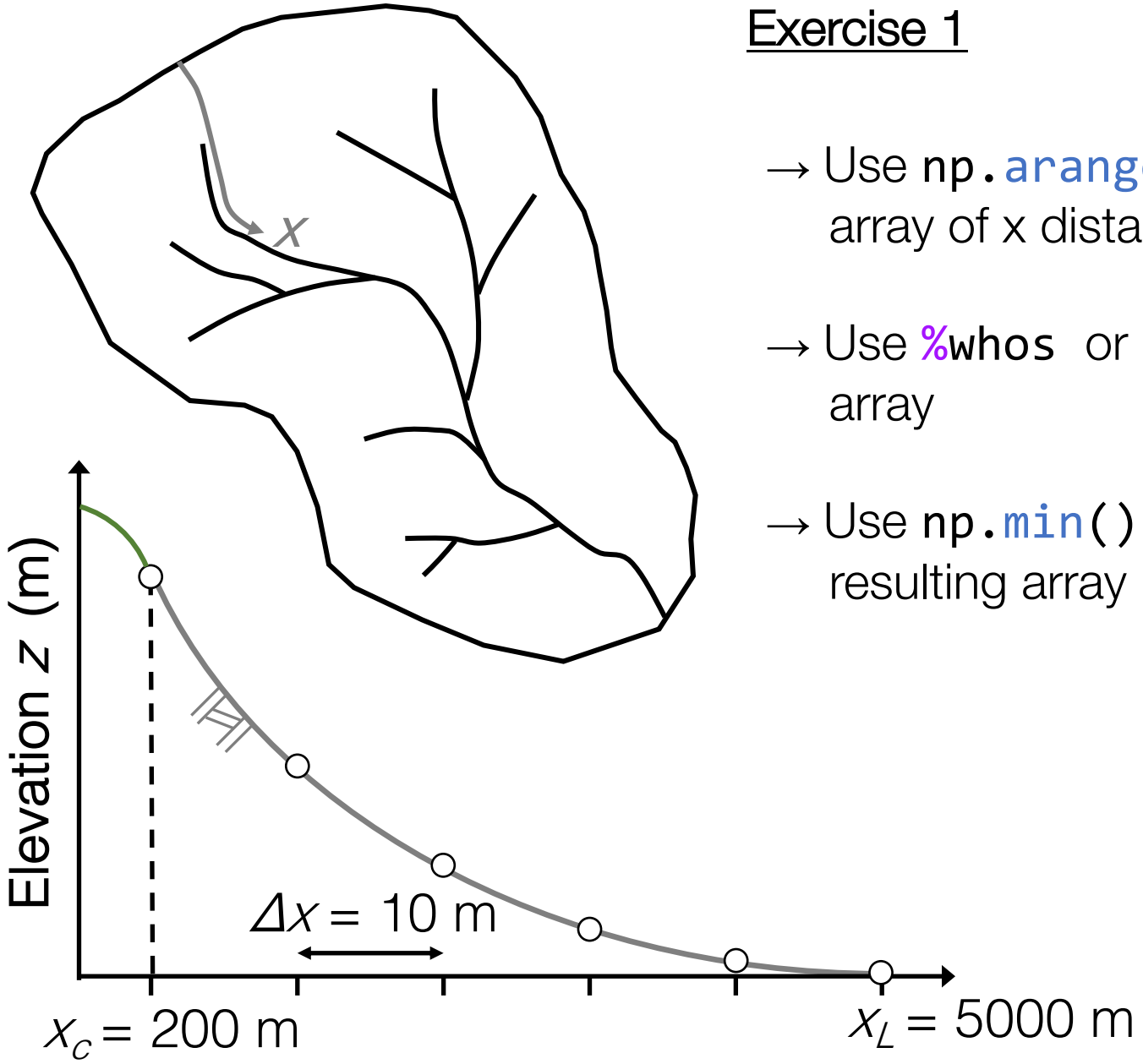
Let's consider a river, initiating at a channel head 200 m from the ridgeline and flowing 4800 m downstream to its outlet. We'll establish a coordinate system with our x-coordinate denoting the streamwise distance from the ridgeline, with the channel head at distance x_c = 200 m and the outlet at distance x_L = 5000 m

Let's start off by defining our streamwise distance coordinates at evenly spaced increments Δx = 10 m.

Use `np.arange()` or `np.linspace()` to create an array of x distance coordinates.

> ⚠ **Tip:** Use e.g., `np.arange?` or `help(np.arange())` to check the documentation of the function – particularly how it handles the end of the interval!

Exercise 1



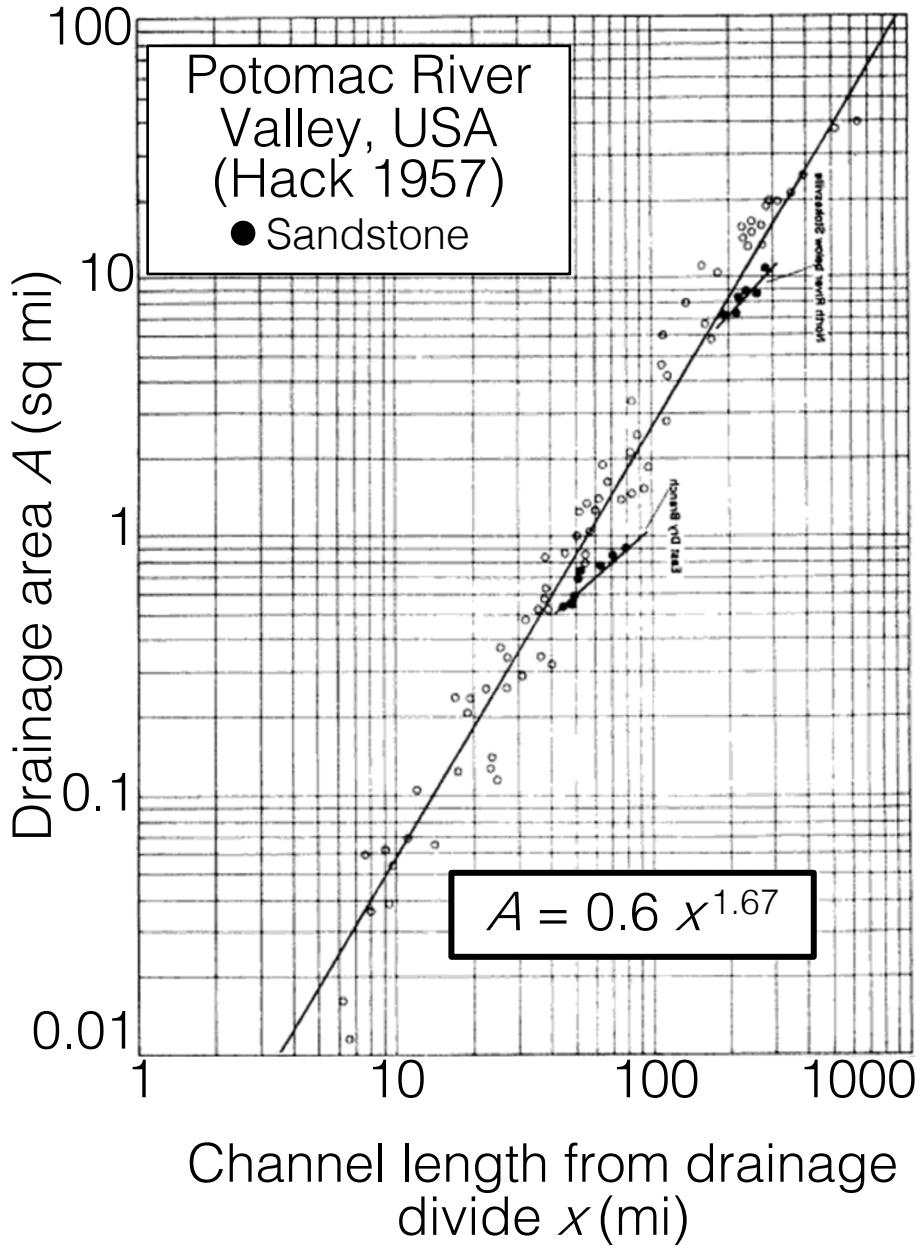
→ Use `np.arange()` or `np.linspace()` to create an array of x distance coordinates

→ Use `%whos` or `len()` to check the length of the resulting array

→ Use `np.min()` or `np.max()` to check the range of the resulting array

Tip: Use e.g., `np.arange?` or `help(np.arange())` to check the documentation of the function – particularly how it handles the end of the interval!

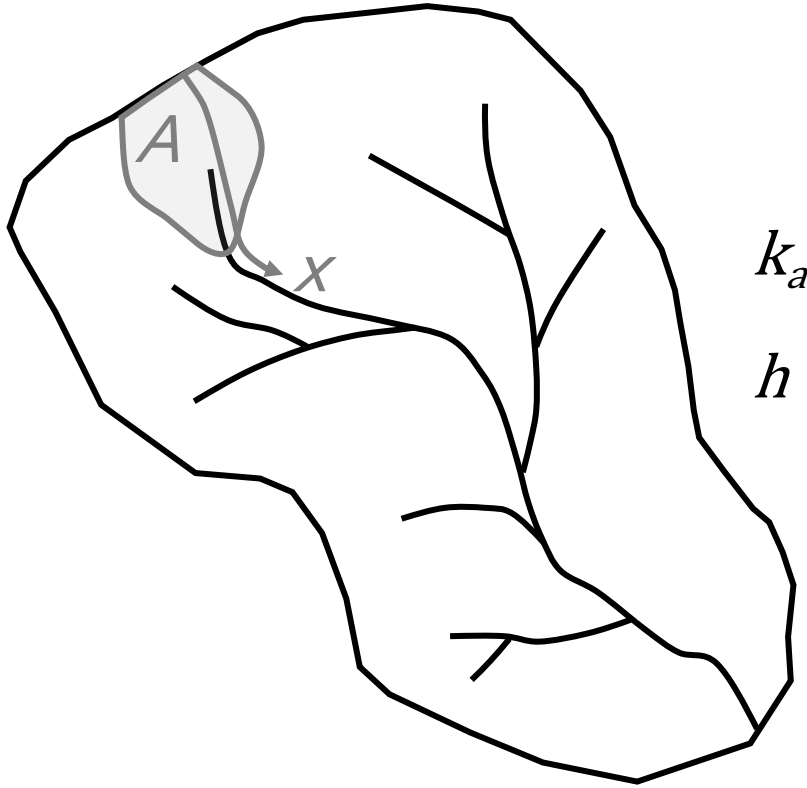
Hack's Law



$$A = k_a x^h$$

Drainage area [L^2]

Streamwise distance from drainage divide [L]



k_a = constant [L^{2-h}]

h = constant…widely observed to be ~1.6 - 2.1

Hack's Law

$$A = k_a x^h$$

Drainage area [L^2]

Streamwise distance from drainage divide [L]

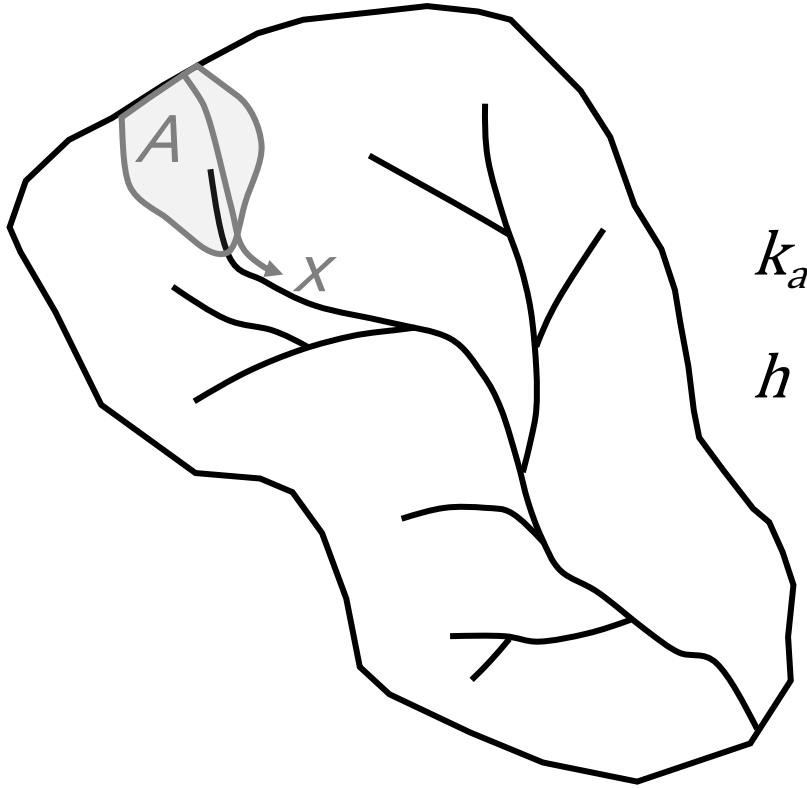
<u>Exercise 2</u>

→ Calculate drainage area along the river profile with…

k_a = 0.5 m $^{2\text{-}h}$

h = 2

→ Plot drainage area vs. streamwise distance from the ridgeline



k_a = constant [$L^{2\text{-}h}$]

h = constant…widely observed to be ~1.6 - 2.1

The stream power law (SPL)

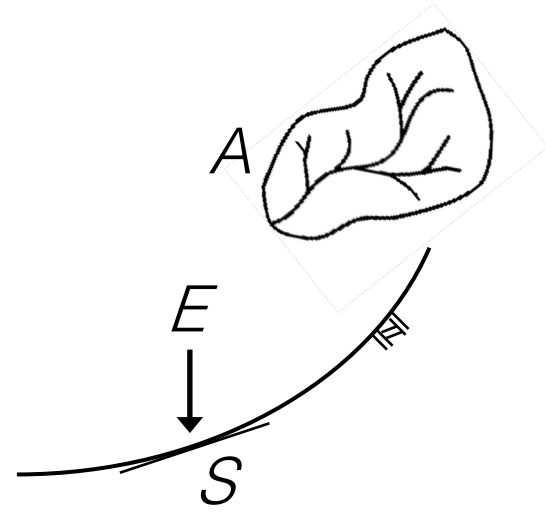
$$E = KA^m S^n$$

Bedrock river incision rate [L T $^{-1}$]

Erodibility = f(rock type, climate,…) [L^{1-2m} T $^{-1}$]

Drainage area (proxy for water discharge) [L^2]

Channel slope [ ]



m = constant = f(hydrology, storm size, …) → typically 0.4 - 0.6

n = constant = f(erosion process) → typically 0.6 – 3.0

The stream power law: Empirical support

Erosion rates from 50 repeat surveys in channel badlands over ~20 years:

$$E = 0.11\ A^{0.44}\ S^{0.68}$$

95% confidence intervals:

$0.06 < K < 0.21$
$0.38 < m < 0.51$
$0.58 < n < 0.78$

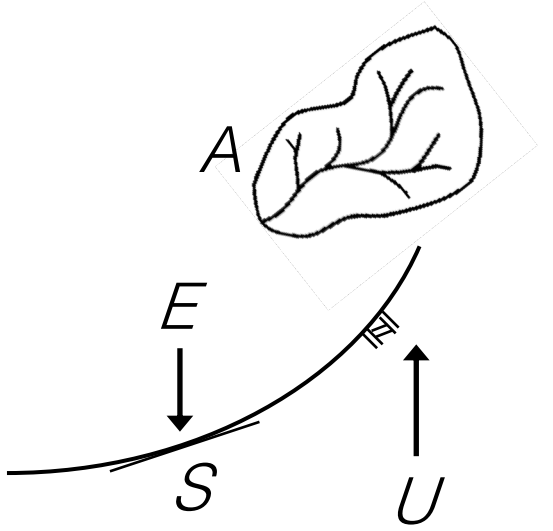
Howard & Kerby 1983

Evolution equation for a longitudinal bedrock river profile

Conservation of mass →

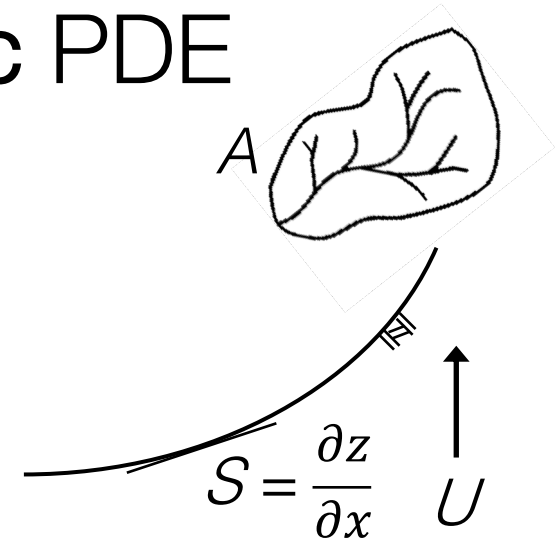
$$\frac{\partial z}{\partial t} = U - E = U - KA^m S^n$$

Rock uplift rate [L T $^{-1}$]



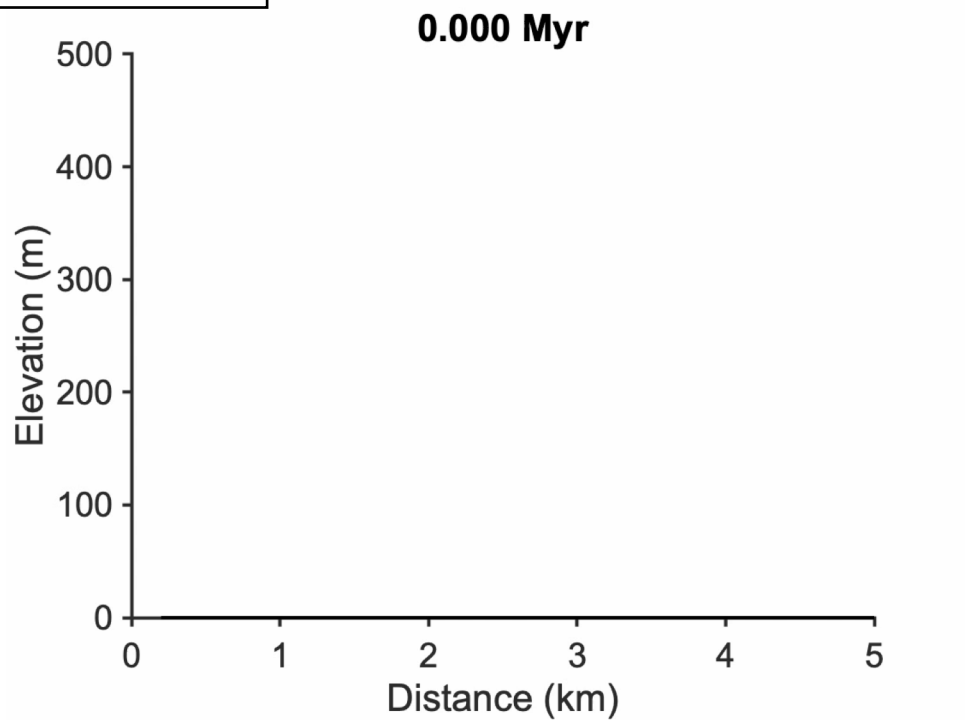
The stream power law is a **hyperbolic** PDE

$$\frac{\partial z}{\partial t} = U - KA^m \left|\frac{\partial z}{\partial x}\right|^n$$

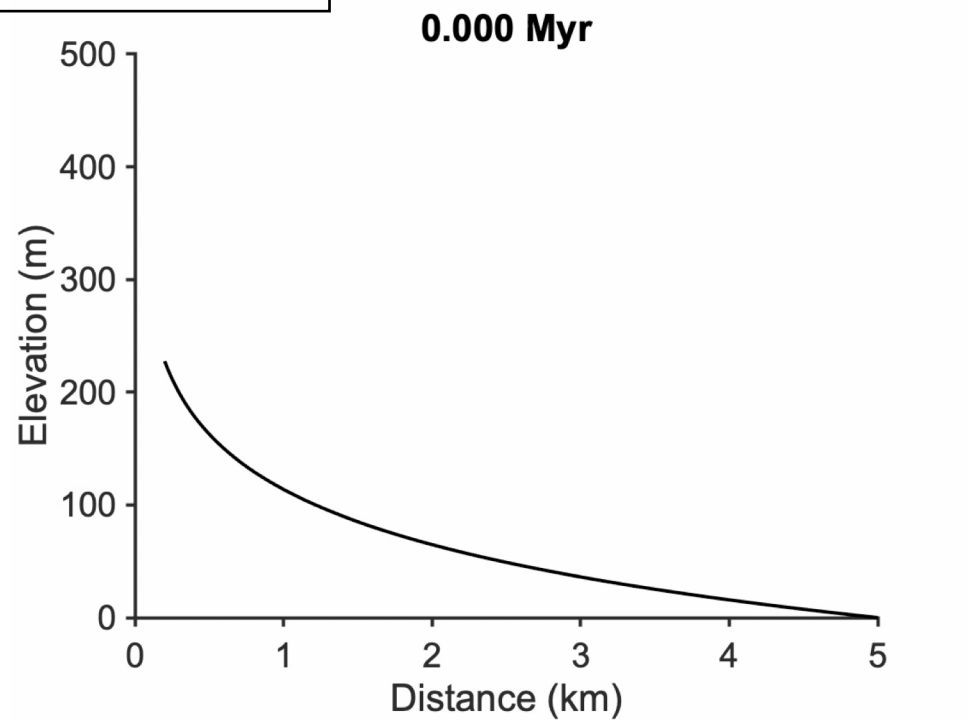


→ Changes in erosion rate driven by climatic or tectonic perturbations propagate upslope at wave speed KA^mS^{n-1}

Plateau uplift



Increase uplift rate



Steady-state longitudinal bedrock river profile

Steady-state: $\frac{\partial z}{\partial t} = 0$

$$0 = U - KA^m \left|\frac{dz}{dx}\right|^n$$

We now have an ordinary differential equation which we can solve analytically.

Rearrange…

$$-\frac{dz}{dx} = \left(\frac{U}{K}\right)^{\frac{1}{n}} A^{-\frac{m}{n}}$$

Steady-state longitudinal bedrock river profile

$$-\frac{dz}{dx} = \left(\frac{U}{K}\right)^{\frac{1}{n}} A^{-\frac{m}{n}}$$

Plug Hack's Law in for A (and define some new variables to simplify)…

$$\frac{dz}{dx} = -\left(\frac{U}{B}\right)^{\frac{1}{n}} x^{-\frac{a}{n}} \quad \text{with } B = K k_a{}^m \quad \text{and} \quad a = hm$$

Separate variables and integrate…

$$\int dz = -\left(\frac{U}{B}\right)^{\frac{1}{n}} \int x^{-\frac{a}{n}}\, dx$$

$$z(x) = -\left(\frac{U}{B}\right)^{\frac{1}{n}} \frac{n}{n-a} x^{\frac{n-a}{n}} + C$$

Steady-state longitudinal bedrock river profile

$$z(x) = -\left(\frac{U}{B}\right)^{\frac{1}{n}} \frac{n}{n-a} x^{\frac{n-a}{n}} + C$$

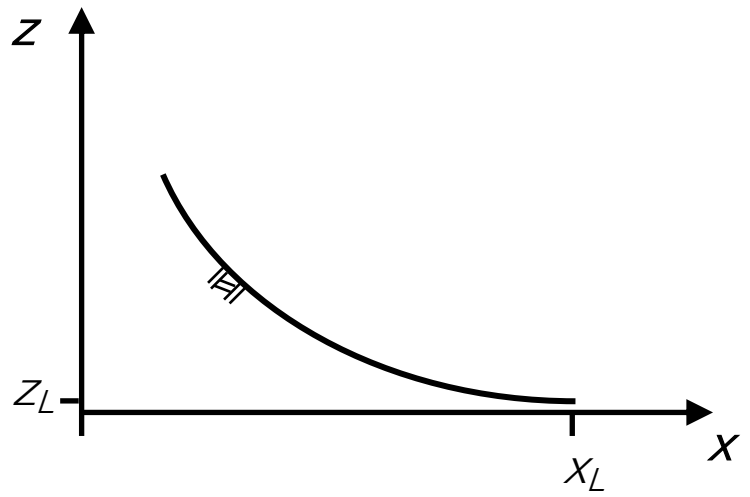
To solve for the integration constant C, plug in boundary condition @ base level: $z(x_L) = z_L$

Case 1: a ≠ n → $C = z_L + \left(\frac{U}{B}\right)^{\frac{1}{n}} \frac{n}{n-a} x_L^{\frac{n-a}{n}}$

$$z(x) = z_L - \left(\frac{U}{B}\right)^{\frac{1}{n}} \frac{n}{n-a}\left(x^{\frac{n-a}{n}} - x_L^{\frac{n-a}{n}}\right)$$

Case 2: a = n → $C = z_L + \left(\frac{U}{B}\right)^{\frac{1}{n}} \ln x_L$

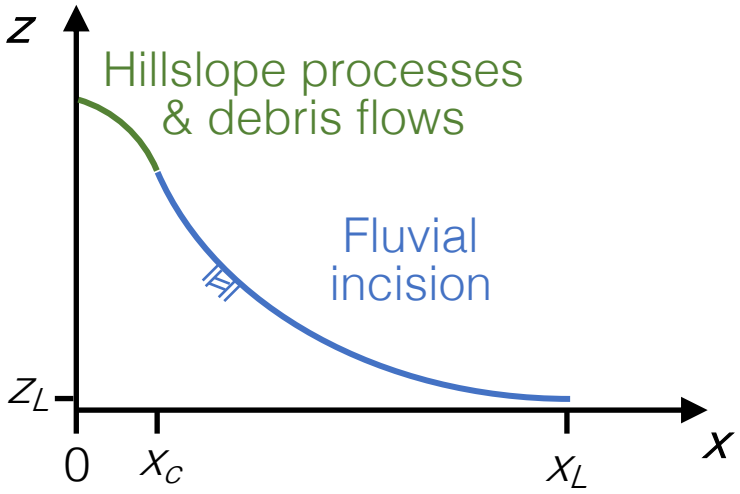
$$z(x) = z_L - \left(\frac{U}{B}\right)^{\frac{1}{n}} \ln \frac{x}{x_L}$$



→ Plug back in $B = Kk_a^m$ and $a = hm$

Steady-state longitudinal bedrock river profile

$$z(x) = \begin{cases} z_L - \left(\frac{U}{K k_a^m}\right)^{\frac{1}{n}} \frac{n}{n-hm} \left(x^{\frac{n-hm}{n}} - x_L^{\frac{n-hm}{n}}\right) & n \neq hm \\ z_L - \left(\frac{U}{K k_a^m}\right)^{\frac{1}{n}} \ln \frac{x}{x_L} & n = hm \end{cases}$$



Note that these solutions are only defined for $x_c \leq x \leq x_L$, where x_c is the distance from the topographic divide where fluvial bedrock channel becomes dominant over hillslope processes and debris flows (typically 200 – 1000 m)

Steady-state longitudinal bedrock river profile

$$z(x) = \begin{cases} z_L - \left(\frac{U}{K k_a^m}\right)^{\frac{1}{n}} \frac{n}{n-hm} \left(x^{\frac{n-hm}{n}} - x_L^{\frac{n-hm}{n}}\right) & n \neq hm \\ z_L - \left(\frac{U}{K k_a^m}\right)^{\frac{1}{n}} \ln \frac{x}{x_L} & n = hm \end{cases}$$

Exercise 3

→ Write a function to calculate the steady–state river profile for any given distance array *x*, outlet elevation z_L, uplift rate *U*, and Hack's Law and SPL parameters

Recall: Defining a function

```
def my_function(x, y):
  return x + y
```

If/else statements

```
if x > y:
    z = x - y
else:
    z = x + y
```

→ Use your function to calculate the steady-state river profile for…

k_a = 0.5 m $^{2-h}$ m = 0.5

h = 2 n = 1

z_L = 0 m

U = 0.001 m/yr

K = 1e-5 m^{1-2m} yr^{-1}

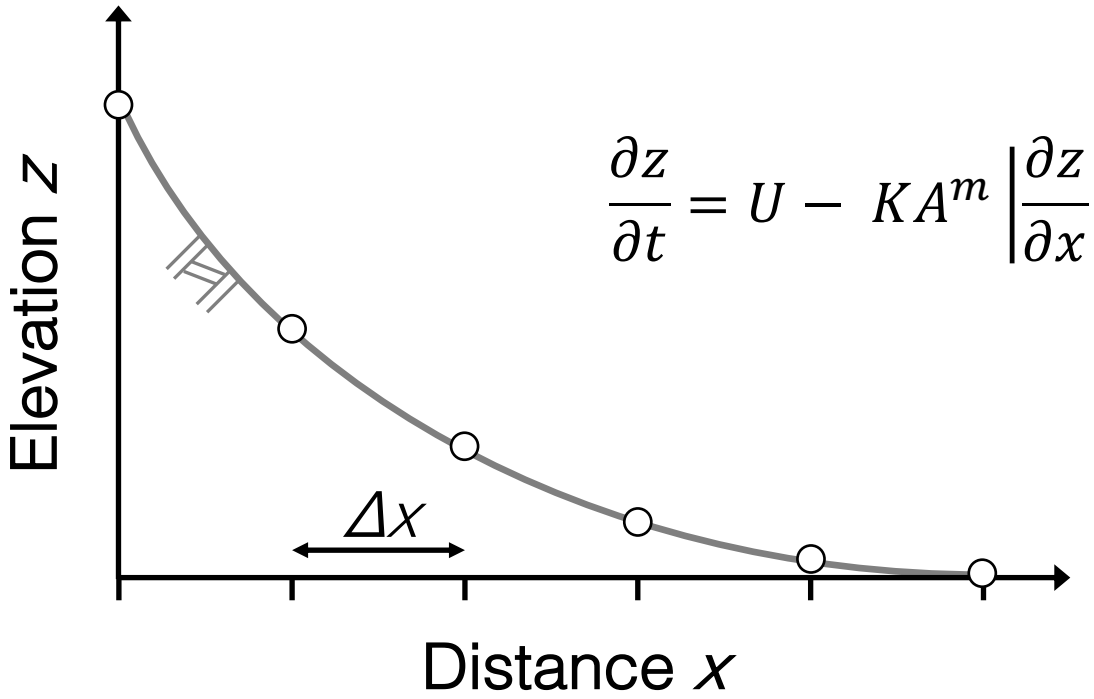
…and plot the results

Finite differencing

What if we are interested in the non-steady-state evolution of a stream profile?

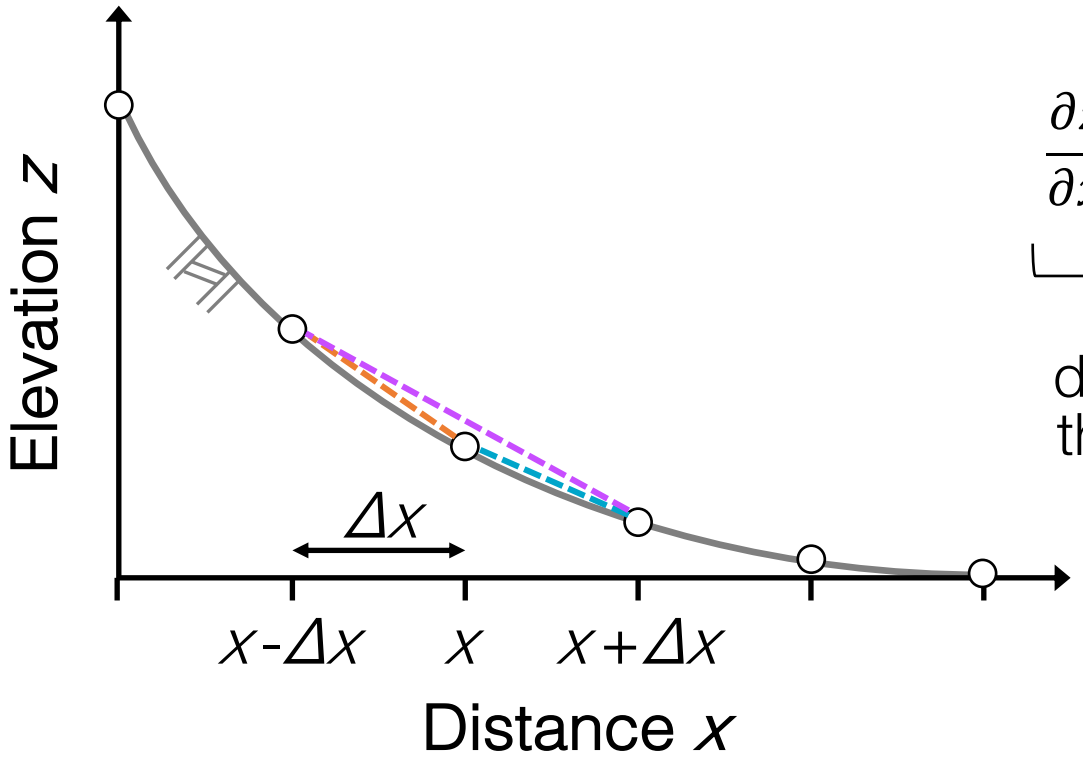
→ It's difficult (though not impossible) to solve non-linear PDEs analytically, so a common approach is to solve them numerically

One simple approach is to replace the continuous partial derivatives with differences over discrete steps in space and time…



$$\frac{\partial z}{\partial t} = U - KA^m \left|\frac{\partial z}{\partial x}\right|^n \longrightarrow \frac{\Delta z}{\Delta t} \approx U - KA^m \left|\frac{\Delta z}{\Delta x}\right|^n$$

Finite difference approximation of the spatial derivative



$$\frac{\partial z}{\partial x} \approx \frac{z(x) - z(x - \Delta x)}{\Delta x}$$

Backward finite difference approx. of the spatial derivative

$$\frac{\partial z}{\partial x} \approx \frac{z(x + \Delta x) - z(x)}{\Delta x}$$

Forward finite difference approx. of the spatial derivative (a.k.a upwind)

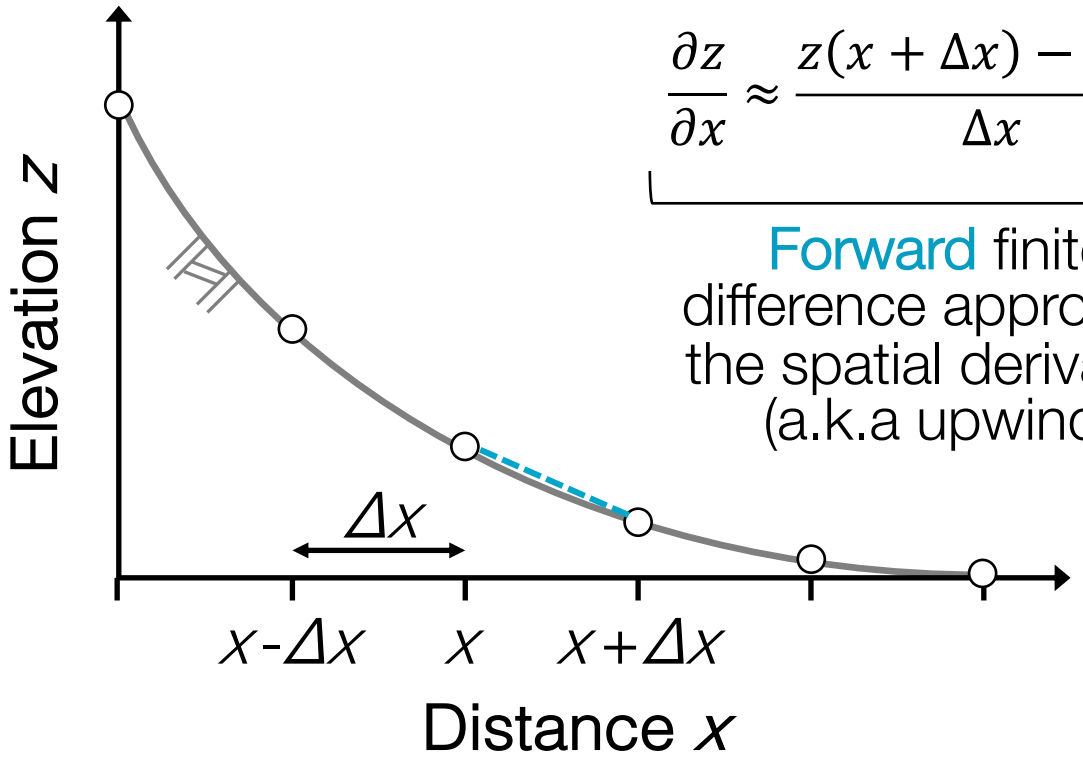
$$\frac{\partial z}{\partial x} \approx \frac{z(x + \Delta x) - z(x - \Delta x)}{2\Delta x}$$

Centered finite difference approx. of the spatial derivative

We'll need **boundary conditions** to define the finite difference approximations:

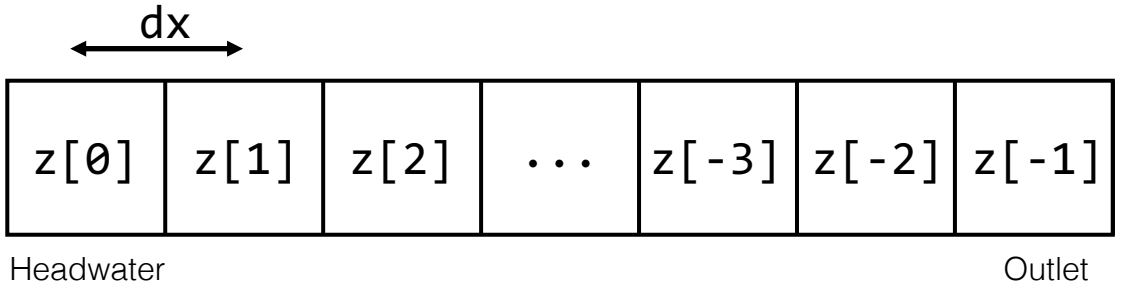
Backward @ the headwater grid point
Centered @ the headwater and outlet grid point
Forward @ the outlet grid point

Finite difference approximation of the spatial derivative



$$\frac{\partial z}{\partial x} \approx \frac{z(x+\Delta x) - z(x)}{\Delta x}$$

Forward finite difference approx. of the spatial derivative (a.k.a upwind)



We can calculate a forward differenced slope at every grid point except `z[-1]` which has no downstream neighbor

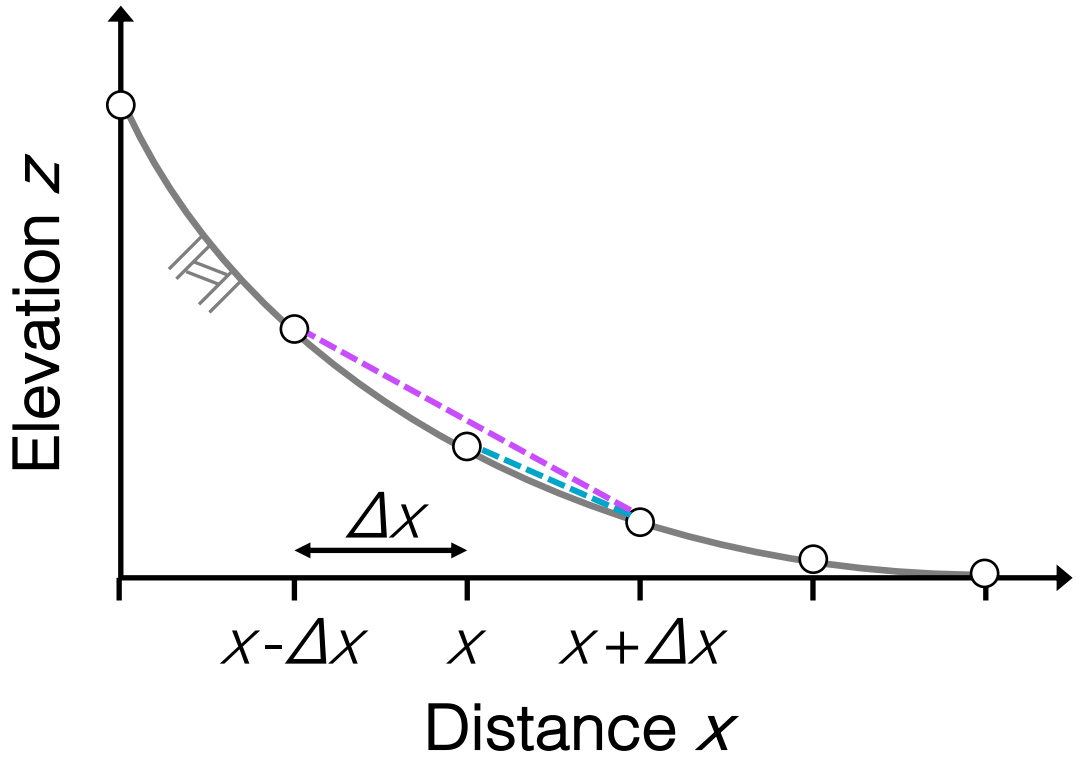
For consistency and later calculations, we'll want to store our slope values in an array the same length as `x` and `z`…so we can first use `np.zeros()` to initialize a slope array `S`

We could then step through this array with a **`for`** loop to calculate the slope at each grid point e.g.

```
for i in range(0,len(z)):
    S[i] = (z[i+1] - z[i])/dx
```

…but in Python there's a more efficient way to do this using array indexing!

Finite difference approximation of the spatial derivative

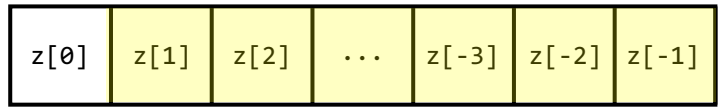


Recall:

We can access all but the first *i* elements of an array using the syntax

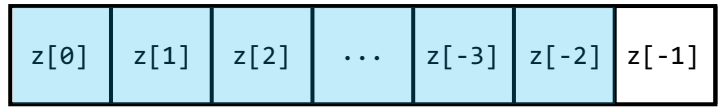
e.g., `i = 1`

`z[i:]`



…or all but the last *i* elements using the syntax

`z[:-i]`



$$\frac{\partial z}{\partial x} \approx \frac{z(x+\Delta x) - z(x)}{\Delta x}$$

Forward finite difference approx. of the spatial derivative

$$\frac{\partial z}{\partial x} \approx \frac{z(x+\Delta x) - z(x-\Delta x)}{2\Delta x}$$

Centered finite difference approx. of the spatial derivative

Exercise 4

→ Using array indexing, write a function that returns the forward and centered space finite difference slope of an input elevation array `z` with distance spacing `dx`