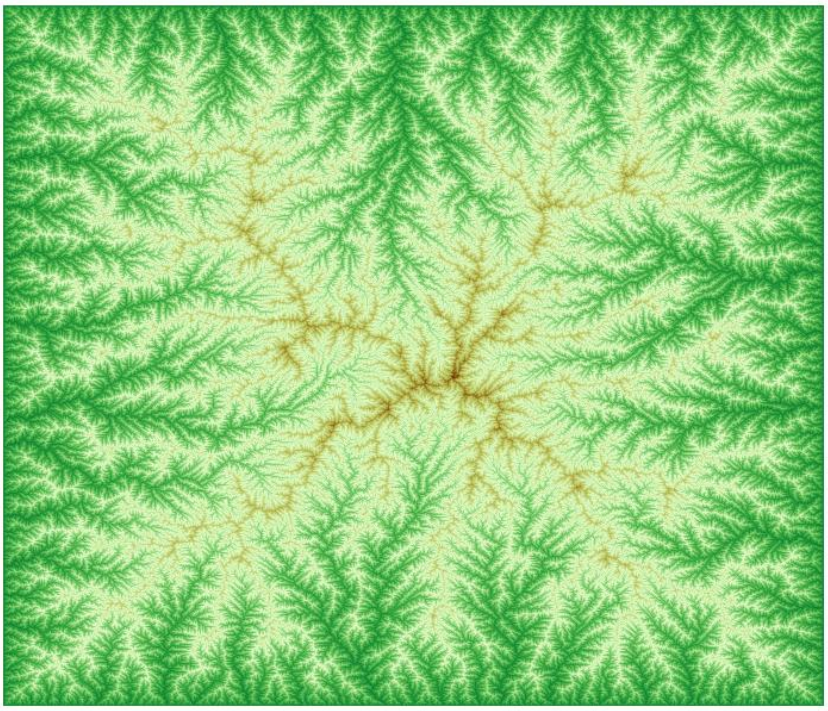

Making a Fluvial Erosion Model

1 Index

2 The (1-D) stream power equation

2.1 Intro/Analytical solutions

The stream power equation describes the geometry and temporal evolution of bedrock streams. By geometry, this means the *long profile*, or the elevation of the river surface going parallel to the direction of the river. An example from the Rio Jáchal, Argentina and its tributaries:

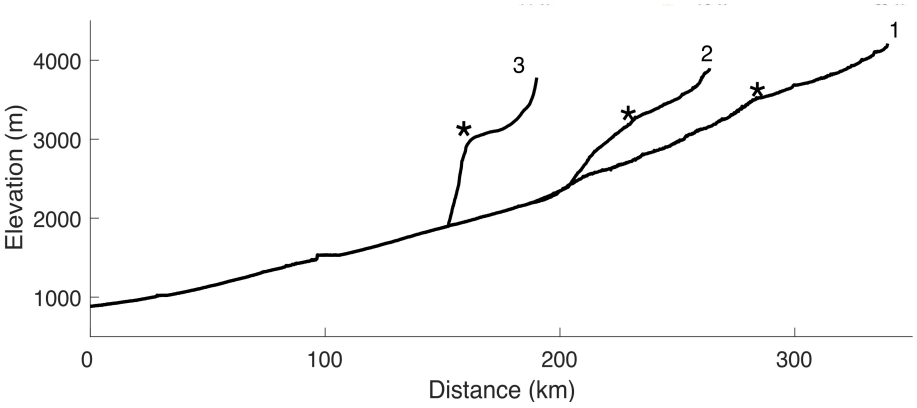

Figure 1: Long river profiles from the Rio-Jáchal, Argentina (1) and tributaries (2 3). Knickpoints, or bumps in the river profile are shown by the asterix. Knickpoints will be discussed later.

Note: Most of what is below is an abbreviated treatment of what can be found in Whipple and Tucker (1999)

The SP equation states that erosion rate within the stream should be proportional to the local slope and the discharge upstream. Typically, upstream drainage area is used as a proxy for discharge. Mathematically it looks like this:

$E = kA^m S^n$

Here A is upstream drainage area and S is local slope. k is called the erodibility, it varies widely and is determined by climate and bedrock, among other things. m and n are exponents which govern the relative importance of slope and drainage area, respectively. The ratio m/n also governs the curvature of the river profile, as we will see. Although the absolute values of m and n are a topic of debate, the ratio m/n is generally around 1/2 in

nature.

The small angle approximation states that at low angles (e.g., most angles of the surface of the earth), $tan(\frac{dz}{dx}) \approx \frac{dz}{dx}$. This means that we can reasonably replace S with $\frac{dz}{dx}$ (local surface gradient).

In the absence of uplift, erosion would solely govern the elevation of the surface so

$$\frac{dz}{dt} = -kA^m(\frac{dz}{dx})^n$$

However, often we want to prescribe some uplift rate (U) that drives landscape change.

$$\frac{dz}{dt} = U - kA^m(\frac{dz}{dx})^n$$

Deriving the steady state solution

Now we can see what the equation looks like for a stream at steady state, that is where rate of surface change is zero: $\frac{dz}{dt} = 0$

$$0 = U - kA^m(\frac{dz}{dx})^n$$

$$\implies (\frac{U}{kA^m})^{1/n}dx = dz$$

$$\implies \int_x^{x_0}(\frac{U}{kA^m})^{1/n}dx = \int dz$$

where x_0 is distance from the headwaters to the river mouth. To integrate we also need to use an approximation for drainage area, which is also a function of downstream distance x. This is given by Hack's law:

$$A(x) \propto x^h$$

Often $h = 2$ is used, and we will see why this is convenient in a moment. Setting the new value of A into the steady state equation above:

$$\int dz = \int_x^{x_0} \frac{U^{1/n}}{k^{1/n} x^{2m/n}} dx$$

$$\implies \text{z(x)} = \frac{U^{1/n}}{k^{1/n}(1 - 2m/n)}(x_0^{1-2m/n} - x^{1-2m/n}), \text{ if 2 m/n} \neq 1$$

or

$$\text{z(x)} = \frac{U^{1/n}}{k^{1/n}}(ln(x_0) - ln(x)), \text{ if } 2m/n = 1$$

Deriving the celerity equation and response time

From the above it is easy to see why $n = 1$, $m = 0.5$, and $h = 2$ is often used, even though there is little evidence that $n = 1$ in nature (although as stated above there is plenty of evidence for $m/n = 0.5$). When the stream profile is perturbed to form a knickpoint (e.g. a waterfall, or some other deviation from steady-state), the velocity at which said knickpoint moves upstream can be determined analytically. Going back to the original equation, this time with $n = 1$:

$$\frac{dz}{dt} = U - kA^m(\frac{dz}{dx})$$

If we divide both sides by $\frac{dz}{dx}$ this becomes:

$$\frac{dx}{dt} = kA^m$$

This is known as a celerity equation. If we assume that Hack's law is true with $h = 2$, then the velocity of the knickpoint will be approximately proportional to its downstream distance.

$$\frac{dx}{dt} = kx^{2m}$$

We can solve for the time it takes for the knickpoint to travel upstream

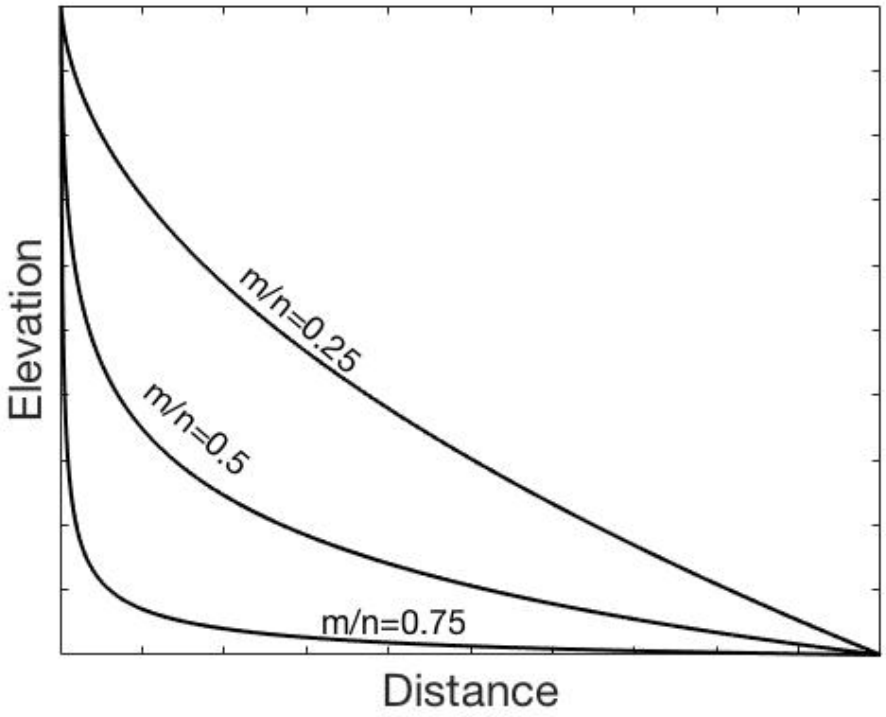

Figure 2: Normalized solutions for the stead state SP equation at different values for the ratio m/n. $m/n \approx 0.5$ is thought to be most common in nature.

by taking the inverse of velocity, and integrating upstream:

$$\int dt = \int_x^{x_0} \frac{1}{kx^{2m}} dx$$

$$\implies t(x) = \frac{1}{k(1-2m)}(x_0^{1-2m} - x^{1-2m}) \text{ , if } 2m \neq 1$$

$$t(x) = \frac{1}{k}(ln(x_0) - ln(x)), \text{ if } 2m = 1$$

Stream Power Equation: $\frac{dz}{dt} = U - kA^m(\frac{dz}{dx})^n$

Parameter	Value
Grid length (L)	100
Grid Resolution (Δx)	1,000 m
Time Step (Δt)	1,000 yr
Hack's exponent (h)	2.0
Drainage Area Exponent (m)	0.5
Slope Exponent (n)	1.0
Erodibility (Bedrock) (k)	1.010^{-6} yr^{-1}
Uplift rate (U)	0.0001 m/yr

Note that the steady state elevation (as derived above) is simply $t(x)U$. We will also see why $n = 1$ is convenient in the numerical model as well.

2.2 Numerical integration: Explicit FD method

Of course there are many conditions for which it is incredibly difficult, often impossible, to integrate the SP equation through time analytically - for example under transient and / or spatially variable uplift. The key is then to integrate using a finite difference method, which in its simplest form can be only a few lines of code in a modern language like MATLAB (all references to programming henceforth will use MATLAB syntax).

First, it is necessary to set up parameters. An example table of parameters is listed below.

Note that we are defining a grid, or (in this 1-D case) a vector, which has a length of 100. This means that we will discretize, or break up the stream's elevation into 100 pieces and solve the stream power equation at each one of these separate pieces.

>>z =zeros(1,L);

The above is MATLAB syntax for creating a vector of length L whose values are all zero. This means that our elevation will start out at zero everywhere. Similarly, the drainage area (as defined by Hack's law), will be broken up into 100 corresponding pieces:

>> A = (delta_x : delta_x: L * delta_x) .^h;

This creates a vector of drainage areas along the entire distance of the stream, in accordance with Hack's law and $h = 2$.

Now we have defined every component on the right hand side of the stream power equation except for $\frac{dz}{dx}$. The most stable way of computing an approximation for $\frac{dz}{dx}$ is known as the upwind difference where the gradient is approximated by the slope between two grid points. If we denote the elevation in the present time step (t) for a given point (i) as z_i^t, then $\frac{dz}{dx} \approx \frac{z_i^t - z_{i+1}^t}{\Delta x}$. Similarly, the left side of the equation can be approximated as the rate of change between time steps: $\frac{dz}{dt} \approx \frac{z_i^{t+1} - z_i^t}{\Delta t}$.
Putting it all together we have:

$$\frac{z_i^{t+1} - z_i^t}{\Delta t} = U - kA^m(\frac{z_i^t - z_{i+1}^t}{\Delta x})^n$$

This can be rearranged to solve for the future time step (t+1) a given grid point (i).

$$z^{t+1} = \Delta t U + z_i^t - \Delta t k A^m(\frac{z_i^t - z_{i+1}^t}{\Delta x})^n$$

Fortunately in MATLAB and other vector based languages the above is just one line:
>> z(1:end-1) = delta_t .* U+z(1:end-1)-k *d elta_t * A(1:end-1) .^m.* ...
(z(1:end-1) - z(2:end)) / delta _x).^n

When we wrap the above statement in a for-loop, we can integrate for as many time steps as we want!

2.3 Implicit solution

The explicit sollution has several disadvantages, but perhaps the biggest one is that it is conditionally stable. This means that if you increase the timestep ,erodibility, or uplift rate by too much, or decrease the grid spacing, it will

explode out of control.

Fortunately, if $n = 1$, it does not take much to implement the implicit solution, which is unconditionally stable. The implicit solution looks similar to the explicit solution, except that we are solving for the gradient using the future time step, as opposed to the current time step:

$$\frac{z_i^{t+1} - z_i^t}{\Delta t} = U - kA^m \frac{z_i^{t+1} - z_{i+1}^{t+1}}{\Delta x}$$

This may seem confusing, to use information from the future time step before we have even solved it, but it should make sense in a moment. Rearranging the equation below to solve for the future time step:

$$z_i^{t+1} = \frac{(U - k\Delta t A^m / z_{i+1}^{t+1} / \Delta x) + z_i^t}{1 + k * A^m * \Delta t / \Delta x}$$

Here we have rearranged the equation so that the right side is only a function of the future timestep at the downstream grid point, and the current timestep at the upstream grid point. This means that if we have a starting point that remains unchanged (e.g., the boundary condition - base level for the stream), then we can solve for the future timestep by successively making our way up-stream. In MATLAB syntax, this would be a simple for-loop:

```
>> f=k*delta_t /delta_x
>> for i=L-1:-1:1
>>        z(i) = (delta_t * U+ f* A(i)^m *z(i+1) + z(i)) / (1+f*A(i)^m);
>> end
```

Again, if we wrap the above in another for-loop we can integrate through time.

3 2-D model, intitial steps

In the 2-D model, the initial steps involve setting up parametrs as in the 1-D model above. Unlike before however, the drainage area (A) is determined implicitly by the flow routing within the model, instead of explicitly

by defining Hack's equation. We can set up the initial 2-D grid using some random function:

>> z = rand(I,J);

The first step to solving the flow routing is to build the stream network by defining some rule by which water flows along the grid. The most simple rule is known as the D-8 rule, which simply says that all the water from each cell in a grid is routed downstream along the path of steepest descent, to one of its 8 neighboring cells.

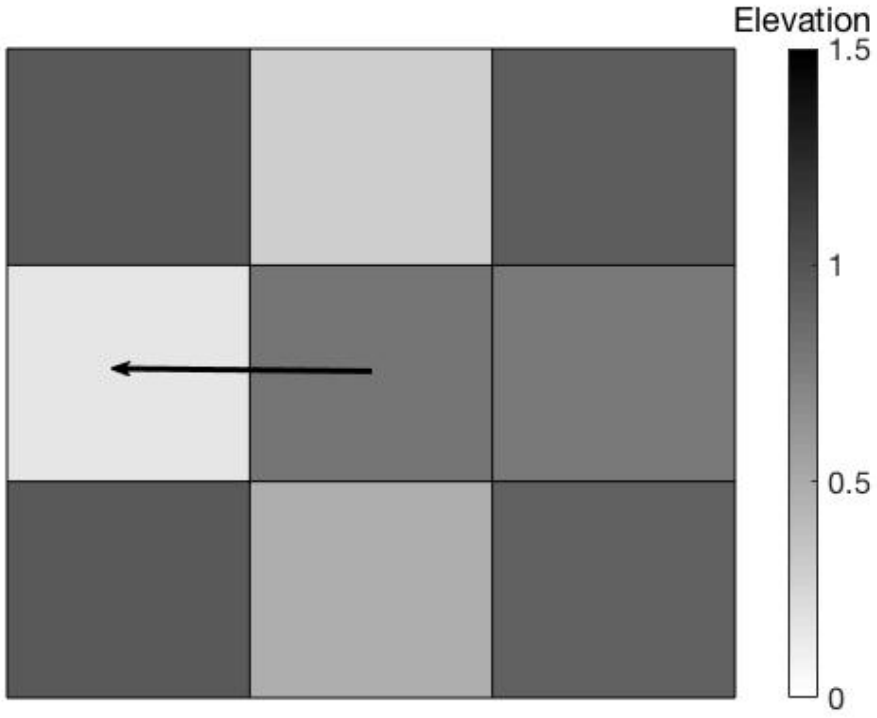

Figure 3: Direction of flowing routing on a rectangular grid determined by the D-8, or steepest descent, rule. Arrow indicates path of the river towards the steepest adjascent cell

To find which cell is the steepest, we can call some sort of simple for loop with an embedded if statement. Keep in mind that we want to loop through

Stream Power Equation: $\frac{dz}{dt} = U + kA^m(\frac{dz}{dx})^n$

Parameter	Value
Grid size (I×J)	100 ×100
Grid Resolution ($\Delta x \times \Delta y$)	1,000 m ×$1,000m$
Time Step (Δt)	1,000 yr
Drainage Area Exponent (m)	0.5
Slope Exponent (n)	1.0
Erodibility (Bedrock) (k)	1.0×10^{-6} yr^{-1}
Uplift rate (U)	0.0001 m/yr

every cell within the matrix, but we do nothing for those on the edge, which are our boundary conditions (this requires some sort of if statment to determine for example, if i>1 and $i < I$. In this case we do the following:

1) Pre-allocate the vector of receivers (call it r), the slopes (call it slopes), distances (call it dxs). In matlab this command is r=zeros(size(z));

2) Loop through all grid points. Initially assign the value of the receiver (r(i,j)) to be itself.You probably want to store the linear index to save space, which is a function of the subindices: $r(i, j) = (j - 1) * I + i$

3) For each grid point, except those on the edge, we also loop through all of its adjacent grid points and calculate the slope between the grid point and its ajdacent grid point.

4) Determine the steepest of the 8 adjacent grid point with slope > 0 (to ensure we are not at a local minima). If at local minima (e.g., slope <= 0), do nothing.

5) When (and if) the steepest neighbor is found, store the index of the receiver. Also store the slope and distance to this point.

6) Repeat, for all grid points.

The above is just an example - you can probably come up with a much

better way of calculating slope yourself, particularly if you use MATLAB's vector features instead of for loops.I should note that although this example is done on a 100×100 grid, it is often much easier to debug your code on a much smaller grid, say 5×5 or even sometimes 3×3. (2×2 would just be boundaries, so don't do that.)

3.1 Creating the stack

The above details the method of finding steepest descent, and therefore the receiver node for every donor node on the grid. Thus we have effectively created connections between nodes which in theory can form a directed graph. Now it is time to sort these nodes based on their connections in a meaningful way, in a structure known as a stack. The stack is designed such that moving in order through the stack, we will never encounter a donor node before we encounter its receiver node, and no node will appear more than once. This can be done in several steps:

1)Pre-allocate the stack (call it s) e.g. s= zeros(1,numel(z));

2) Start with an edge node or local sink node (whose slope is still zero from the above computation). You can find these with one line from the slope stack in MATLAB: b=find(slopes==0); You may have to convert the linear indices back to subindices i=mod(b−1,I)+1; j=floor((b−1)/I)+1; Add it to the end of the stack (the first available 0 value on the vector s).

3) Loop through and determine if each neighbor node is a donor to the current node. If so, add the coordinates of this node to the end of the stack.

4) Repeat (3) for the next node in the stack. Go until you reach the end of the stack (which will eventually stop growing if done properly, I promise).

5) Repeat (2) for the next edge node.

Note that this stack is slightly different than the one describe by Braun and Willett (2013), but its function is the same. In my opinion, this one is simpler because it does not require use of a recursive function, and instead simply uses a while loop.

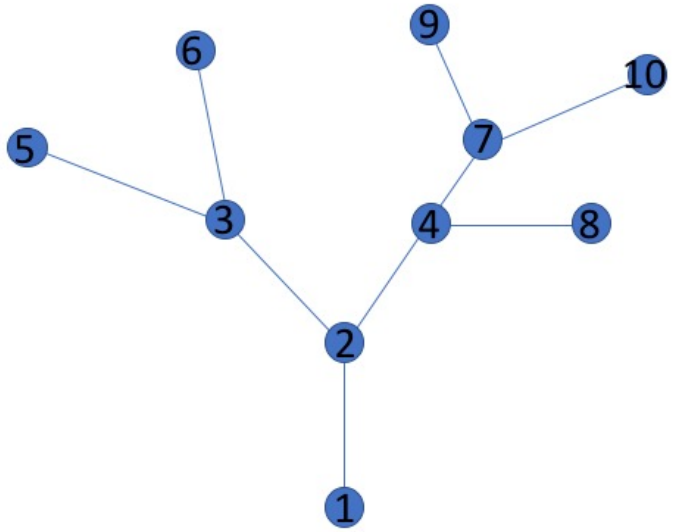

Figure 4: The ordering of the stack for a given set of points in a drainage network

3.2 Calculating Drainage area

Now that we have the stack it is extremely simple, and quick, to calculate A. First, generate a matrix A of the same size as z, where the values in each cell correspond to its drainage area contribution (typically $\Delta x \times \Delta y$). Then, cumulatively sum the drainage area from each donor onto each receiver in in the reverse order of the stack:

```
>>A=zeros(size(z))+delta _x × delta _y.
>>for l=length(stacki):-1:1
>>        A(r(s(l)))=A(r(s(l)))+A(s(l));
>>end
```

3.3 Calculating Erosion

Now here is the time to put your skills to the test - see the above section on the 1-D implicit method and apply it using the stack. Remember, we move in the forward order of the stack this time (upstream).

References

Braun, J. and Willett, S. D. (2013). A very efficient O(n), implicit and parallel method to solve the stream power equation governing fluvial incision and landscape evolution. *Geomorph.*, 180:170–179.

Whipple, K. X. and Tucker, G. E. (1999). Dynamics of the stream-power river incision model: Implications for height limits of mountain ranges, landscape response timescales, and research needs. *Journal of Geophysical Research: Solid Earth*, 104(B8):17661–17674.