Python for scientific computing: An unstructured adaptive cubed sphere algorithm for non-singular mesh generation in spherical coordinates

Federico Gatti

April 12, 2021

MOX – Modelling and Scientific Computing
Dipartimento di Matematica, Politecnico di Milano
Piazza Leonardo da Vinci, 20133 Milano, Italy
federico.gatti@polimi.it

In this report we describe an unstructured adaptive cubed-sphere algorithm able to produce a mesh in spherical coordinates without singularities at the poles. The described algorithm has been implemented in python mainly using *numpy* and *scipy* for vector handling and interpolation purposes and the *gmsh* python API, see [2], to perform all meshing tasks. This project is intended to provide a tool able to generate a non-singular unstructured grid as input to a meteorological numerical simulation. The adaptation procedure is driven by the orography function, in particular the estimator is based on the H^1-seminorm of the orography function.
The report is structured as follows, in Section 1 we introduce the reader to the mesh generation problem in spherical coordinates and to the cubed sphere algorithm, in Section 2 we briefly describe the space-adaptation procedure here adopted, finally in Section 3 we present some results and provide some implementation details.

1 The cubed sphere algorithm

In meteorological applications one issue is the generation of a non-singular mesh in spherical coordinates. Indeed, standard meshing algorithms fail to produce an efficient mesh from the computational standpoint because of its exceedingly high resolution at higher latitudes (the longitudinal resolution approaches zero at the poles). In this work we use the *equidistance cubed-sphere* algorithm. This method was first introduced by [4] to produce a quasi-uniform mesh on a globe for atmospheric models. The method is a composition of six *central gnomonic transformation*, a single gnomonic projection is able to represent appropriately less than one-sixth of the planet [5]. In the following, we briefly describe the classical cubed-sphere method, we refer to [3] for a deeper explanation. The idea of the method is to perform a projection of the interested spherical slices

onto a proper set of cube faces via a proper gnomonic projection transformation, then to perform the meshing via standard meshing algorithms and finally back-transform to the spherical geometry.
Let us refer to Figure 1, we consider a sphere with radius R and a cube inscribed to the sphere with edge length $2a$ where $a = \frac{R}{\sqrt{3}}$. The cube is oriented such that the global cartesian coordinate axes (X, Y, Z) are orthogonal to the cube faces $P_n, n = 1, \dots, 6$. The link between the longitude and latitude spherical coordinates (λ, θ) and the absolute cartesian coordinates (X, Y, Z) on the sphere reads,

$$\begin{cases} X = R\cos\lambda\cos\theta, \\ Y = R\sin\lambda\sin\theta, \\ Z = R\sin\theta, \end{cases} \tag{1}$$

where (λ, θ) are such that $\lambda \in [-\pi, \pi)$ and $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

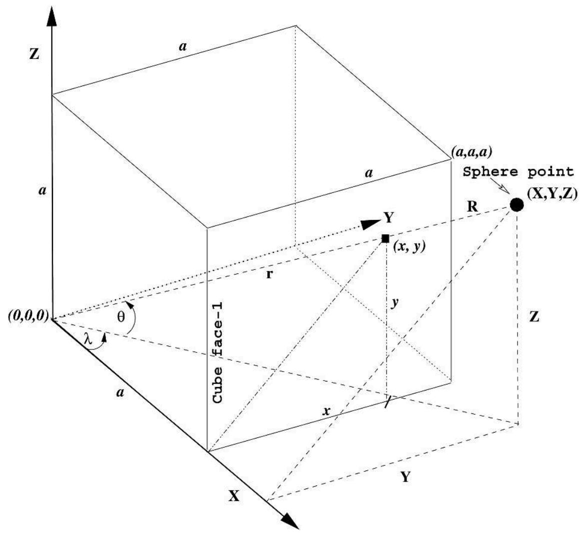


Figure 1: Schematic representation of the gnomonic central transformation, note that only one-eighth of the cube is represented. Figure taken from [3].

Let (x, y) be the local Cartesian coordinates centered on the surface of each cube face such that $x, y \in [-a, a]$. Considering $r = \sqrt{a^2 + x^2 + y^2}$, the gnomonic projection on each cube face P_n reads,

- On P_1, i.e. for $\cos\lambda \geq \frac{1}{\sqrt{2}}$ and $|\theta| \leq \arctan\cos\lambda$,

$$\begin{cases} x = a\tan\lambda, \\ y = a\tan\theta\sec\lambda. \end{cases} \tag{2}$$

  The global triplet (X, Y, Z) and the local coordinates (x, y) are then coupled as,

$$(X, Y, Z) = \frac{R}{r}(a, x, y).$$

- On P_2, i.e. for $\sin\lambda \geq \frac{1}{\sqrt{2}}$ and $|\theta| \leq \arctan\sin\lambda$,

$$\begin{cases} x = -a\cot\lambda, \\ y = a\tan\theta\csc\lambda, \end{cases} \tag{3}$$

$$(X,Y,Z) = \frac{R}{r}(-x,a,y).$$

- On P_3, for $\cos\lambda \leq -\frac{1}{\sqrt{2}}$ and $|\theta| \leq -\arctan\cos\lambda$,

$$\begin{cases} x = a\tan\lambda, \\ y = -a\tan\theta\sec\lambda, \end{cases} \tag{4}$$

$$(X,Y,Z) = \frac{R}{r}(-a,-x,y).$$

- On P_4, i.e. for $\sin\lambda \leq -\frac{1}{\sqrt{2}}$ and $|\theta| \leq -\arctan\sin\lambda$,

$$\begin{cases} x = -a\cot\lambda, \\ y = -a\tan\theta\csc\lambda, \end{cases} \tag{5}$$

$$(X,Y,Z) = \frac{R}{r}(x,-a,y).$$

- On P_5, i.e. for ($\cos\lambda \geq \frac{1}{\sqrt{2}}$ and $\theta \geq \arctan\cos\lambda$) or ($\sin\lambda \geq \frac{1}{\sqrt{2}}$ and $\theta \geq \arctan\sin\lambda$) or ($\cos\lambda \leq -\frac{1}{\sqrt{2}}$ and $\theta \geq -\arctan\cos\lambda$) or ($\sin\lambda \leq -\frac{1}{\sqrt{2}}$ and $\theta \geq -\arctan\sin\lambda$),

$$\begin{cases} x = a\sin\lambda\cot\theta, \\ y = -a\cos\lambda\cot\theta, \end{cases} \tag{6}$$

$$(X,Y,Z) = \frac{R}{r}(-y,x,a).$$

- On P_6, i.e. for ($\cos\lambda \geq \frac{1}{\sqrt{2}}$ and $\theta \leq -\arctan\cos\lambda$) or ($\sin\lambda \geq \frac{1}{\sqrt{2}}$ and $\theta \leq -\arctan\sin\lambda$) or ($\cos\lambda \leq -\frac{1}{\sqrt{2}}$ and $\theta \leq \arctan\cos\lambda$) or ($\sin\lambda \leq -\frac{1}{\sqrt{2}}$ and $\theta \leq \arctan\sin\lambda$),

$$\begin{cases} x = -a\sin\lambda\cot\theta, \\ y = -a\cos\lambda\cot\theta, \end{cases} \tag{7}$$

$$(X,Y,Z) = \frac{R}{r}(y,x,-a).$$

We want now to introduce some metric notions for the central projections that will be useful for the adaptation procedure described in next section. Let us consider a scalar continuous function b, e.g. an orography function, as function of (λ,θ), the gradient in spherical coordinates reads,

$$\nabla b = \frac{1}{R\cos\theta}\partial_\lambda b\ \hat{\lambda} + \frac{1}{R}\partial_\theta b\ \hat{\theta}, \tag{8}$$

where $\hat{\lambda}$, $\hat{\theta}$ are the unit base vectors of longitude and latitude coordinates. Considering the base vectors as function of the spatial position vector $\mathbf{r} = x\,\hat{x} + y\,\hat{y}$, i.e.,

$$\hat{\theta} = \partial_\theta \mathbf{r}, \quad \hat{\lambda} = \partial_\lambda \mathbf{r}, \tag{9}$$

one can define, with an abuse of notation, in the base unit vectors $\hat{x}, \hat{y}$ of the local cartesian coordinate system, the gradient as,

$$\nabla b = \begin{pmatrix} \frac{1}{R}\partial_\theta b & \frac{1}{R\cos\theta}\partial_\lambda b \end{pmatrix} \mathbf{A} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix}, \tag{10}$$

where the sphere-to-cube transformation matrix $\mathbf{A}$ is defined for each central projection and reads,

$$\mathbf{A} = \begin{pmatrix} \partial_\theta x & \partial_\theta y \\ \partial_\lambda x & \partial_\lambda y \end{pmatrix}. \tag{11}$$

2 Space-adaptation

We perform the mesh-adaptation via the *size function*, that specifies the desired size of the elements of the computational mesh starting from a given baseline mesh.

Let us consider the discrete input orography b_h^* defined on a regular triangulation $\mathcal{T}_h^*$ on Ω in the (λ, θ) coordinate space. The approximation space is the linear finite element space $\mathbb{P}_1$, where $\mathbb{P}_1 \subset H^1(\Omega)$. The adaptation procedure is driven by the H^1-seminorm of the discretization error, considering the solution b_h defined on a baseline triangulation $\mathcal{T}_h$, the discretization error wrt the starting solution b_h^*, reads,

$$|e|^2_{H^1(\Omega)} = \int_\Omega |\nabla b_h^* - \nabla b_h|^2 \; dA, \tag{12}$$

where $b_h = \Pi_h^1 b_h^*$, being Π_h^1 the piecewise linear interpolator. We integrate the discretization error on the baseline mesh $\mathcal{T}_h$ using the midpoint numerical integration, if $\mathbf{x}_m$ is the geometrical center of the K-th element we have,

$$|e|^2_{H^1(\Omega)} = \sum_{K \in \mathcal{T}_h} |G_h(\mathbf{x}_m) - \nabla b_h|^2 \, |K|, \tag{13}$$

where $G_h = \Pi_h^0 \nabla b_h^*$, being Π_h^0 the piecewise constant interpolator. Note that the interpolated orography gradient G_h belongs to $\mathbb{P}_0 = \nabla \mathbb{P}_1$, and sometimes in the following we refer to this quantity as slope.

We describe now the procedure to obtain an *isotropic metric* from the estimator presented above. A metric is a function that prescribes the size of the elements of the mesh, so it defines the size field. In particular, we compute for each $K \in \mathcal{T}_h$ a scalar value h_K which provide the target diameter of the new element. Let η^2 describe the error estimator defined in (13),

$$\eta^2 = \sum_{K \in \mathcal{T}_h} \eta_K^2, \tag{14}$$

if we denote with τ a given tolerance on the H^1-seminorm of the discretization error (in this project we set $\tau = 0.1$), assuming the error is equidistributed, i.e.

that each element K contributes with the same quantity to the estimator, we get,

$$\eta_K^2 = \frac{\tau^2}{N}, \tag{15}$$

where N is the number of elements of the current mesh. Finally, defining the scaled quantity $\tilde{\eta}_K^2 = \frac{\eta_K^2}{|K|} = \frac{\eta_K^2}{h_K^2|\hat{K}|}$, where $\hat{K}$ is a reference triangle, and combining this expression with (15) we obtain the size field,

$$h_K = \left(\frac{\tau^2}{|\hat{K}|\tilde{\eta}_K^2 N}\right)^{\frac{1}{2}}. \tag{16}$$

3 Numerical results

In our project b_h^* is given by a slice of the GTOPO30 [1], i.e. a global raster digital elevation model (DEM) providing terrain elevation data with a horizontal grid spacing of 30 arc seconds (approximately 1 kilometer), we import the dataset with the help of the *netcdf4* python interface. In the following, we have set $a = 1$ so $R = \sqrt{3}$.

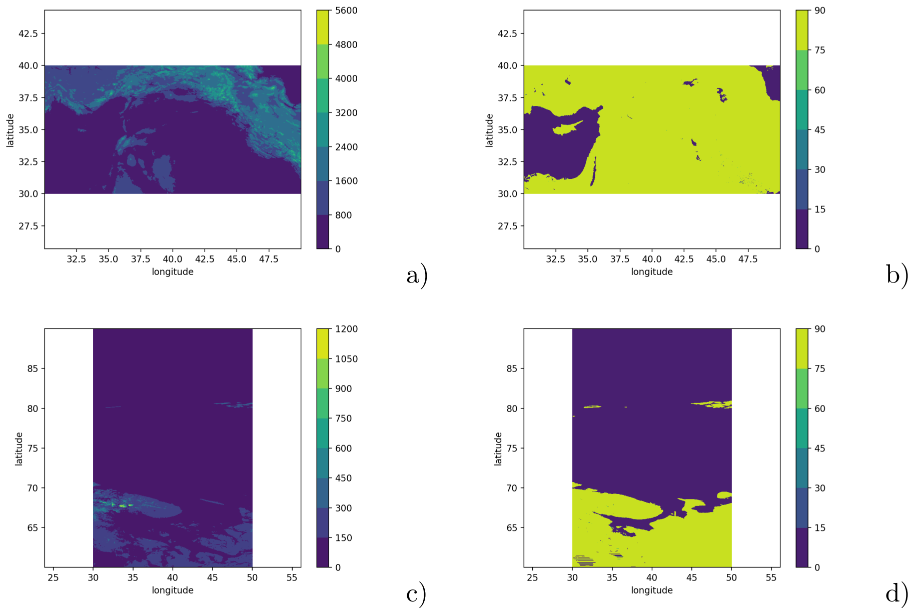
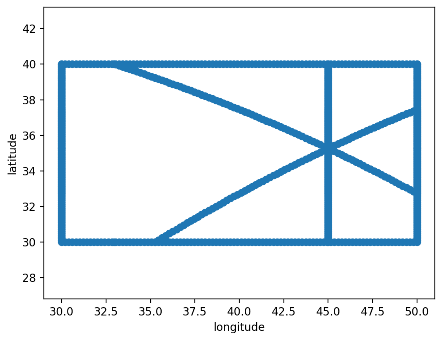
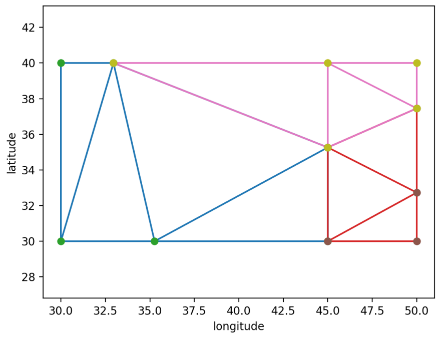


Figure 2: Isolines in meters of b_h^* in the case study domain a) $\Omega = (30°, 50°) \times (30°, 40°)$ and c) $\Omega = (30°, 50°) \times (60°, 90°)$. b), d) isolines in degrees of the modulus of the starting slope ($\arctan |\nabla b_h^*|$).

In a first example, we want to show the ability of the implementation to provide a conformal mesh even at the edges of the cube. Let us take a spherical slice domain given by $\Omega = (30°, 50°) \times (30°, 40°)$ with the corresponding b_h^* as in Figure 2a), b). To ensure conformity at the cube edges, we need to compute all possible intersection points of each cube face boundary curve among each other and among the boundary of $\partial\Omega$, e.g. the delimiting curves of the face P_1 are $\cos\lambda = \frac{1}{\sqrt{2}}$ and $|\theta| = \arctan\cos\lambda$ (refer to Equations from (2) to (7)). Via image processing logical operations we obtain the intersection points, then we perform a triangulation of this set via *scipy* Delaunay taking care to have conformity at the interface between cube faces, see Figure 3. In this case we discover that Ω is divided among three cube faces, P_1, P_2, P_5. For each vertex of each triangle represented in Figure 3b) we take the corresponding image in the corresponding cube face, we parametrize each triangle edge with a straight line (except for edges belonging to $\partial\Omega$ which are straight lines in the (λ, θ) domain so parametrized with a spline in the cubed domain) and then perform triangulation of the whole piece-wise plane surface, i.e. the union of the whole region Ω in the cubed domain.



a)



b)

Figure 3: a) In blue we delineate the curves having intersection point inside Ω, b) In blue we have the triangulation on P_1, in red the triangulation on P_2 and in pink the triangulation on P_5. Note that in the (λ, θ) coordinate system the triangle is not a cartesian triangle, i.e. the link between vertices is not in general a straight line, here we represent the link as a straight line being interested only in the vertices themselves.

The meshing is performed via *gmsh* python API and leads to the baseline mesh $\mathcal{T}_h$ and corresponding solution b_h obtained via piece-wise linear interpolating the solution b_h^* in the points of the baseline mesh. Note that the interpolation operations (both piece-wise linear and constant for respectively orography and slope) are performed in the (λ, θ) coordinate system, with the help of regular

grid interpolation scipy routine we can perfomr this rather fastly compared to standard scipy interp2d taking advantage of the regular structure. Note that we need also to multiply by the sphere-to-cube matrix transformation (11) to obtain the correct gradients, as described in Section 1. The characteristic size of the baseline mesh should be set fine enough to prevent to filter out orography being our estimator based on the L^2-norm of the slope and integrated with simple midpoint integration rule. In this work we have set this roughly equal to the starting spatial resolution at the equator divided by 5. This choice is justified together with the a-priori choice to have as minimum resolution of the final mesh one order of magnitude lower than the starting spatial resolution at the equator. Indeed in the compute size field routine we have set this quantity as the finest resolution and the coarsest resolution equal 20 times the initial resolution at the equatorial line.
Once computed the size field, we put this in the gmsh python API to produce a new cartesian mesh in the cubed domain. We report in Figure 4 the results both in the cube domain and in the (λ, θ) domain.

Finally, to report the ability of the current implementation to produce a non-singular mesh even at high latitudes, we consider the input data in Figure 2c), d) i.e. with $\Omega = (30°, 50°) \times (60°, 90°)$. Note that in the cubed domain this involves only the P_5 face. Following the same steps underlined above for the previous example and with the same baseline mesh parameters, we obtain the final mesh as in Figure 4c), d). Note that the back-transformed mesh, i.e. in the spherical coordinates, looses isotropy more and more approaching the poles while being non-singular. A possible remedy could be the implementation of an anisotropic mesh refinement estimator in the cubed domain, depending on the latitude.

4 Conclusions

We have presented a python preprocessor able to produce an unstructured non-singular adaptive mesh that can be included in a numerical simulation tool for meterological applications. The implementation is based on *gmsh* to produce the mesh and can provide both triangular and quadrilateral mesh.
This project can be easily extended implementing other projection transformations, e.g. *equiangular cubed-sphere* projection, or can be further improved considering the coastal line, in particular, the projection of the set of coastal lines onto the cube to produce a mesh conformal to this line.

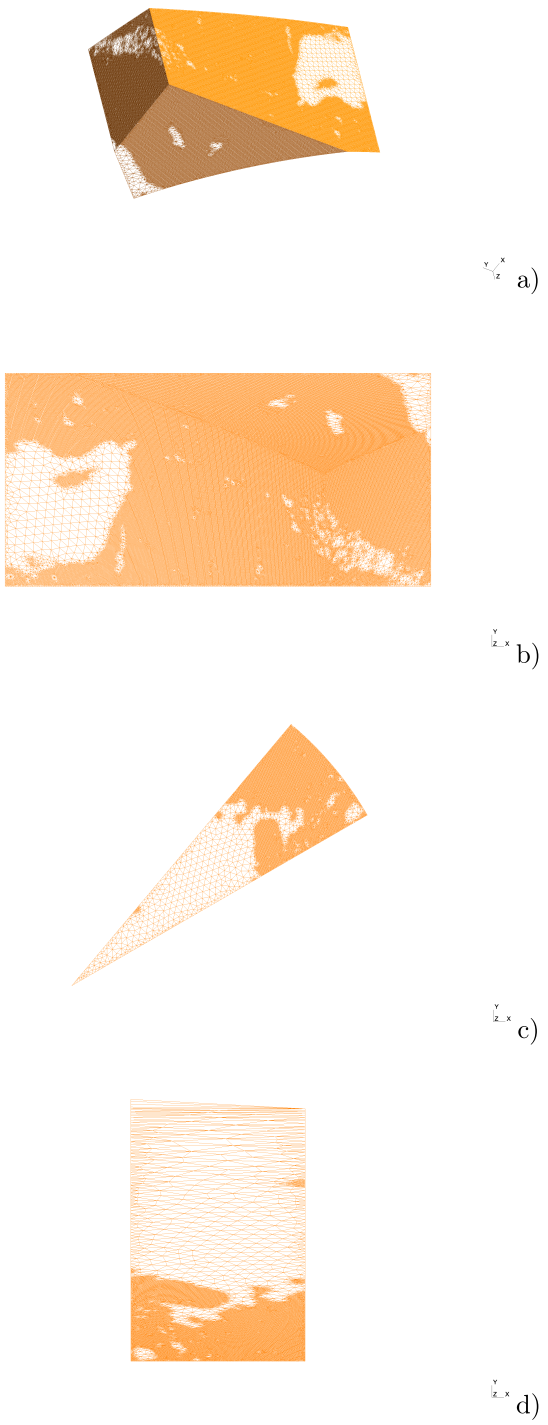


Figure 4: a), c) Final mesh in the cubed domain. b), d) Final mesh in the (λ, θ) coordinate system, note that the coarsest zones correspond flat places, see Figure 2b), mainly sea/water places.

References

[1] Earth Resources Observation and Science Center, U.S. Geological Survey, U.S. Department of the Interior. Usgs 30 arc-second global elevation data, gtopo30, 1997.

[2] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[3] Ramachandran D Nair, Stephen J Thomas, and Richard D Loft. A discontinuous galerkin transport scheme on the cubed sphere. *Monthly Weather Review*, 133(4):814–828, 2005.

[4] Robert Sadourny. Conservative finite-difference approximations of the primitive equations on quasi-uniform spherical grids. *Monthly Weather Review*, 100(2):136–144, 1972.

[5] John Parr Snyder and Philip M Voxland. *An album of map projections*. Number 1453. US Government Printing Office, 1989.