Coordinate system

The earth is represented as the S_2 unit sphere immerged in $\mathbb{R}^3$. Coordinates are $\{\theta, \phi\}$, where $\theta \in [0, 2\pi)$ spans longitudes eastwardly and $\phi \in \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right]$ spans latitudes from south to north pole. The immersion follows the standard application:

$$\begin{array}{ccc} [0, 2\pi) \Diamond \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right] & \longrightarrow & S_2 \subset \mathbb{R}^3 \\ \{\theta, \phi\} & \longrightarrow & X_{(\theta,\phi)} = \begin{pmatrix} \cos(\theta)\cos(\phi) \\ \sin(\theta)\cos(\phi) \\ \sin(\phi) \end{pmatrix} \end{array}$$

As computations are perfomed in degrees, minutes and seconds, the usual scaling $\alpha \longrightarrow \frac{\pi}{180}\alpha$ is introduced, thus providing the following expression:

```
In[222]:= X[{θ_, ϕ_}] := {Cos[π/180 θ] Cos[π/180 ϕ], Sin[π/180 θ] Cos[π/180 ϕ], Sin[π/180 ϕ]};
```

Orthodromy

An orthodromy is a shortest route linking two locations on a geometrical surface, according to the intrisic metric of that surface. Orthodromies are usually computed following a extremal principle delivering a second order differential equation, whose solution is the orthodromy. On the sphere, othodromies are constructed as follow: consider two locations $\{\theta_1, \phi_1\}$ and $\{\theta_2, \phi_2\}$on the sphere. Together with the center of the sphere, $X_c = \{0, 0, 0\}$, the points $X_{(\theta_1,\phi_1)}$ and $X_{(\theta_2,\phi_2)}$ define a plane whose intersection with the sphere is a circle on the sphere, indeed the orthodromy. Of course, starting from $\{\theta_1, \phi_1\}$, $\{\theta_2, \phi_2\}$ can be reached by travelling on the circle into two, shorter or longer, directions.

Historically, orthodromies on earth are called “arcs of great circle”, whereas the arc is indeed the short path.

Making use of the scalar product between two vectors A and B: $A \cdot B = \|A\| \, \|B\| \, \mathrm{Cos}[\sphericalangle(A, B)]$, the distance on the sphere between two locations $\{\theta_1, \phi_1\}$ and $\{\theta_2, \phi_2\}$ is provided by the module **Ortho** computing $\mathrm{ArcCos}[X_{(\theta_1,\phi_1)} . X_{(\theta_2,\phi_2)}]$ (remember that the radius of our sphere is unity). Making use of // FullSimplify provides the following module, delivering the distance between two locations, expressed in nautical miles.

```
In[223]:= Ortho[{x1_, y1_}, {x2_, y2_}] := Module[{θ1, ϕ1, θ2, ϕ2},
    {θ1, ϕ1, θ2, ϕ2} = π/180 {x1, y1, x2, y2};
    N[(360 60)/(2 π) ArcCos[Cos[θ1 - θ2] Cos[ϕ1] Cos[ϕ2] + Sin[ϕ1] Sin[ϕ2]]]
  ];
```

Why nautical miles?

On earth, one Nautical Mile is the distance separating two locations $\{\Theta_1, \phi_1\}$ and $\{\Theta_2, \phi_2\}$ making with the centre of earth X_c an angle $\sphericalangle(X_{(\Theta_1,\phi_1)}, X_c, X_{(\Theta_2,\phi_2)})$ that equals to one minute of degree.
This explains the coefficient $\frac{360 \times 60}{2\pi}$ in the expression of **Ortho**. By that way, our computations are performed without any reference to the actual radius of earth, or of any planet on which a regatta would be organized. Furthemore, computing speed in knots - nautical miles per hour - makes us only dependent on the angular rotation of the planet, one full rotation of 2 π in 24 hours. This definition justifies the way the module **Ortho** is concieved. This system is valid on any other planet or spherical body. Expressed in our usual unit system, the values of the nautical mile and the knot would simply be different. And of course, I almost forgot it, on earth 1 Nautical Mile = 1.852 km, 1 Knot = 0.514 m/s.

Constructing an Array of Nodes

The following module **Graphe** enables the construction of a grid of nodes, indeed the array of nodes onto which a decision tree will be afterwards computed. All elements are given in $\{\theta, \phi\}$ = {longitudes, latitudes}, in this order. Two anchors are provided as input, AncrDep for Anchor at Dearture and AncrArr for Anchor at Arrival. Represented as green dots on following figure 1, they enable the positionning of the grid and are provided as parameters $\{\Theta_1, \phi_1\}$ and $\{\Theta_2, \phi_2\}$ at the call of the module Graphe. The number of nodes in the array is given by {m, n}. The number of steps from departure to arrival is specifiend by m, in that case 20, the number of options by n, running between -n and +n, in that case 21 options. The parameter ρ specifies the ratio of the distances between two steps and two options, The euclidian distance between Departure and Arrival is computed as $e = \sqrt[2]{(\Theta_2 - \Theta_1)^2 + (\phi_2 - \phi_1)^2}$. The gridsteps are given by $\frac{e}{m}$ and $\rho\,\frac{e}{m}$. As the graph can be rotated in any fashion, the rotation angle τ is provided by Cos[τ] $= \frac{(\Theta_2-\Theta_1)}{e}$ and Sin[τ] $= \frac{(\phi_2-\phi_1)}{e}$,

thus delivering the rotation matrix $M_{[\tau]} = \begin{pmatrix} \text{Cos}[\tau] & -\text{Sin}[\tau] \\ \text{Sin}[\tau] & \text{Cos}[\tau] \end{pmatrix}$. The construction of the array of nodes is finally provided by: $\begin{pmatrix} \text{Cos}[\tau] & -\text{Sin}[\tau] \\ \text{Sin}[\tau] & \text{Cos}[\tau] \end{pmatrix} \cdot \begin{pmatrix} \frac{e}{m}\, i \\ \rho\,\frac{e}{m}\, j \end{pmatrix} + \begin{pmatrix} \Theta_1 \\ \phi_1 \end{pmatrix}$,

as expressed in the module **Graphe**.

```
In[224]:= Graphe[{θ1_, ϕ1_}, {θ2_, ϕ2_}] := Module[{ρ, e, c, s, M, m, n},
    ρ = 0.3;
    {m, n} = {5, 2};
    e = Sqrt[(θ2 - θ1)^2 + (ϕ2 - ϕ1)^2];
    {c, s} = {(θ2 - θ1)/e, (ϕ2 - ϕ1)/e};
    M := ( c  -s
          +s   c );
    Table[N[M.{e/m i, ρ e/m j} + {θ1, ϕ1}], {i, 0, m}, {j, -n, n}]];
```

Example

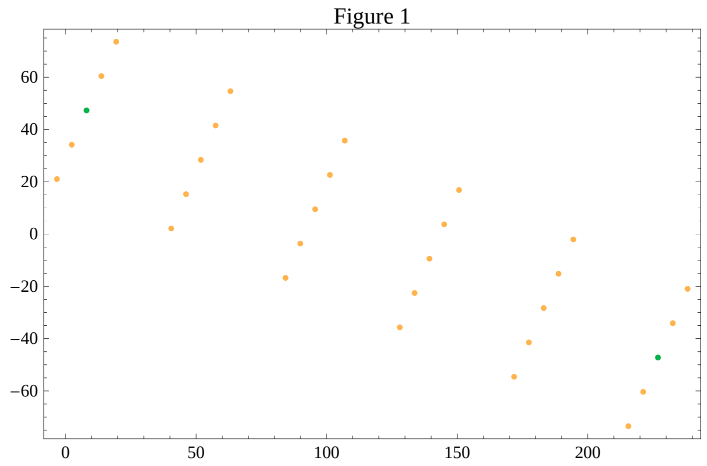
Figure 1 shows the array of nodes constructed with the module Graphe with a Departure anchor located in Northeast Switzerland, an Arrival Anchor in South Pacific. Achors **AncreDep** and **AncreArr** are {longitudes, latitudes} defined in degrees in the two first instructions, they are pictured in green in figure 1. The matrix **Loc** contains the coordinates {θ, ϕ} of the nodes in the array. It is constructed by the call **Loc = Graphe[AncreDep, AncreArr].** The dimension of **Loc** is then computed with the instruction **{Maxi, Maxj, Coord}=Dimensions[Loc]. Loc, Maxi** and **Maxi** will be used later by the module DynProgr while performing the dynamical programming.

```
In[225]:= AncreDep = {8.05, 47.3};
          AncreArr = {226.88, -47.22};
          Loc = Graphe[AncreDep, AncreArr];
          {Maxi, Maxj, Coord} = Dimensions[Loc];
```

Following graphical instructions draw figure 1.

```
In[229]:= GD = Graphics[{RGBColor[0, 0.7, 0.3], Point[AncreDep]}];
          GA = Graphics[{RGBColor[0, 0.7, 0.3], Point[AncreArr]}];
          G1 = Table[Graphics[{RGBColor[1, 0.7, 0.3], Point[Loc[[i, j]]]}], {i, 1, Maxi}, {j, 1, Maxj}];
          Show[G1, GD, GA, Frame → True, PlotLabel → "Figure 1"]
```

Out[232]=



Dynamical Programming, a first glance

Dynamical Programming covers a set of methodolgies aimed at optimizing sequential decisions. Belonging to the realm of applied mathematics, it consist in defining a sequence of steps at which decisions have to be settled, taking into account progress done so far, as well as future opportunities. Invented by the end of the second World War, Dynamical Programming has blossomed into a vast array of fields, from logistics to finance.

In our application - sailing -, the objective is obvious: defining the course of a sailing boat enabling the shortest time from a departure to an arrival point. To that purpose, geographical specifications, as well as winds ahead on water surfaces and their interaction with the sailing boat have to be integrated into a comprehensive decision scheme.

Getting started

Let we start smoothly and consider firstly a vehicle moving on earth's surface at constant speed. It is for example required to travel from the Departure to the Arrival Achor in the previous diagramme. The sequencing is provided by the slicing of the array on figure 1, from the northwest to the southeast side of the array. Starting with a very simple formulation that will be progressively refined, one defines:

$\triangle t_i$, i = 1 ... **m**, is the time requiered to travel from the slice i-1 to the slice i, with i = 0 corresponding to the first slice, to which Departure Anchor belongs, and i = **m** for the last slice, to which Arrival Anchor belongs.

D_i, i = 0 ... m the time at which the slice i is crossed

T_i, i = 0 is the departure time,

T_i, i = m is the arrival time

Our optimal programme can thus be defined as "minimize the travel duration D_m from slice 0 to slice m":

$$D_m = \underset{i=1}{\overset{m}{\text{Min}}}\left[\sum_{i=1}^{m} \triangle t_i\right]. \qquad (1)$$

Let we consider an intermediary step I and evaluate $D_I = \underset{i=1}{\overset{I}{\text{Min}}}\left[\sum_{i=1}^{I} \triangle t_i\right]$. It can be rewritten:

$D_I = \underset{i=1}{\overset{I}{\text{Min}}}\left[\triangle t_I + \sum_{i=1}^{I-1} \triangle t_i\right] = \overset{I}{\text{Min}}\left[\triangle t_I + \underset{i=1}{\overset{I-1}{\text{Min}}}\left[\sum_{i=1}^{I-1} \triangle t_i\right]\right] = \overset{I}{\text{Min}}[\triangle t_I + D_{I-1}]$. Our initial question emerges anew in the the penultimate expression at step I-1. Thus, equating the first and final expressions provides a recursive formulation of the problem. An initial value has to be specified, provided by $D_0 = D_{i=0}$ = departure time, giving:

$$D_I = \overset{I}{\text{Min}}[\triangle t_I + D_{I-1}]; \qquad D_0\text{: departure time.} \qquad (2)$$

This derivation results from a profound optimization principle, assessing that a strategy is optimal if and only if all its sub-strategies are optimal. Of course, such a flat rate formulation conceals the decision process.

Deciding consist here in choosing:

for each node j of the slice I-1, towards which node k of the next slice I the boat should sail,

or equivalently,

for each node k of the slice I from which node j of the previous slice I-1 the boat should have sailed.

Introducing the latter condition in the previous expression for D_I gives (with standard {i, j} typography for representing nodes in graph theory):

$$D_{\{I,k\}} = \underset{j=-n}{\overset{j=n}{\mathrm{Min}}}\left[\triangle t^{\{I,\,k\}}_{\{I-1,\,j\}} + D_{\{I-1,j\}}\right]; \qquad D_0\text{: departure time.} \qquad (3)$$

The node j^*belonging to the slice I-1 and satisfying the above minimum is the decision taken in the slice I for the node k. It is expressed as:

$$D_{\{I,k\}} \longrightarrow j^* \in \text{ slice I-1.} \qquad (4)$$

The reader will have noticed that the expression D - for Duration - can be easly traded for D, for Decision. Adopting this convention, D_I in expression (2) is the set of all the decisions taken in the slice I, for each node k of that slice:

$$D_I \longrightarrow \{j^*_{-n} \;...\; j^*_k \;...\; j^*_n\} \subset \text{ slice I-1.} \qquad (5)$$

To be remembered, the number of steps from departure to arrival is specifiend by **m**, the number of options by **n**, running between -n and +n, in that case 21 options.

A further refinement has to be introduced. For a node located on one side of the array, as for example in the i-th slice on the left side of the array, node $\{i,\,-n\}$, the algorithm could choose the node $\{i-1,\,+n\}$ as its predecessor, thus forcing a lateral crossing through the array, indeed an unfortunate situation. A control mechanism, defining for each node {i, k} in the i slice which are the {i-1, k} nodes that can be addressed in the i-1 slice, is introduced. It is controlled by the ***Spread*** parameter whose modus operandi is defined as:

$$D_{\{I,k\}} \longrightarrow j^* \text{ with } |j^* - k| \leq \text{Spread}; \; j^* \in \text{ slice I-1.} \qquad (6)$$

A neighborhood is thus defined in slice i-1 for each node k in slice i. Other nodes $\{i-1,\,j^*\}$ not satisfying the requirement $|j^*$ - k| ≤ Spread are simply disregarded during the selection process. In a further version of the algrorithm, the spread parameter will enable us to cope with the geography of the lake, when the array of nodes may reach outside of its shore. Everything said above is encapsulated in the following Dynamical Programming Module, that deserves to be yet described.

The Module DynProgr

DynProgr performs the dynamical programming and delivers the matrix **Decision** pictured below. This matrix describes all relevant parameters with each of its elements representing one node of the decision tree. It has dimensions **Maxi**, **Maxj**, the dimension of the table of coordinates **Loc** delivered by the module **Graphe** previously presented. **DynProgr** uses an internal module **ProgrDyn** that will be discussed below.

DynProgr starts after the comment line (* DynProgr starts here ;*). In the first instruction, a matrice **Grph** is generated with all its elements initialized at $\begin{pmatrix} 0 \\ 1\,000\,000 \end{pmatrix}$. The 0 is an empty location that will be used later, the 1’000’000 is a navigation duration, expressed in hours and initializated at a high value: 1 million hours. The starting node, given as input of the module: *Start_*, is determined in the second instruction: The locator is set at one, the departure time at 0 hours. One notices that the starting node is not necessarily the Departure Anchor, it can be set on any node of the first row. The third instruction executes the dynamical programming: it computes expression (5) from the second slice (indexed **r**) up to the last one: **Do[ProgrDyn[r], {r, 2, Maxi}]**. The transposition finally provides the matrix **Decision** exhibited below.

ProgrDyn, the internal module, receives as input parameter the current slice and do exactly three things:

1. Instruction **Etabli**: computes for each node **{slice, k}** the table of arrival times at that node **{slice, k}** from each node **{slice-1, j}**, using the arrival time at node **{slice-1, j}** given by **Grph[[slice-1, j, 2]]** and adding **Ortho[Loc[[slice-1, j]],Loc[[slice, k]]]. Loc[[slice-1, j]]** and **Loc[[slice, k]]** are $\{\theta, \phi\}$ of both nodes.

2. Instruction **{{Posi}}** uses a standard *Mathematica* command to find the position of a minimum value in the table **Etabli,** thus giving us j^*_k in expression (5).

3. Last instruction computes the spread, as defined in expression (6) and updates the matrix **Grph.**

This concludes the description of DynProgr.

```
In[244]:= DynProgr[Start_] := Module[{Grph, ProgrDyn},

    ProgrDyn[slice_] := Module[{Spread, Etabli, Posi},
      Spread = 4;
      Do[
           Etabli = Table[Grph[[slice - 1, j, 2]] + Ortho[Loc[[slice - 1, j]], Loc[[slice, k]]], {j, 1, Maxj}];

           {{Posi}} = Position[Etabli, Min[Etabli]];

           If[Abs[k - Posi] ≤ Spread, Grph[[slice, k]] = {{{slice - 1, Posi}, {slice, k}}, Min[Etabli]}], {k, 1, Maxj}]

          ]; (* end ProgrDyn *)

    (* DynProgr starts here ;*)

    Grph = Table[{0, 10^6}, {i, 1, Maxi}, {j, 1, Maxj}];

    Grph[[1, Start]] = {1, 0};

    Do[ProgrDyn[r], {r, 2, Maxi}];

    Transpose[Grph]

  ]; (* End Programmation Dynamique *)
```

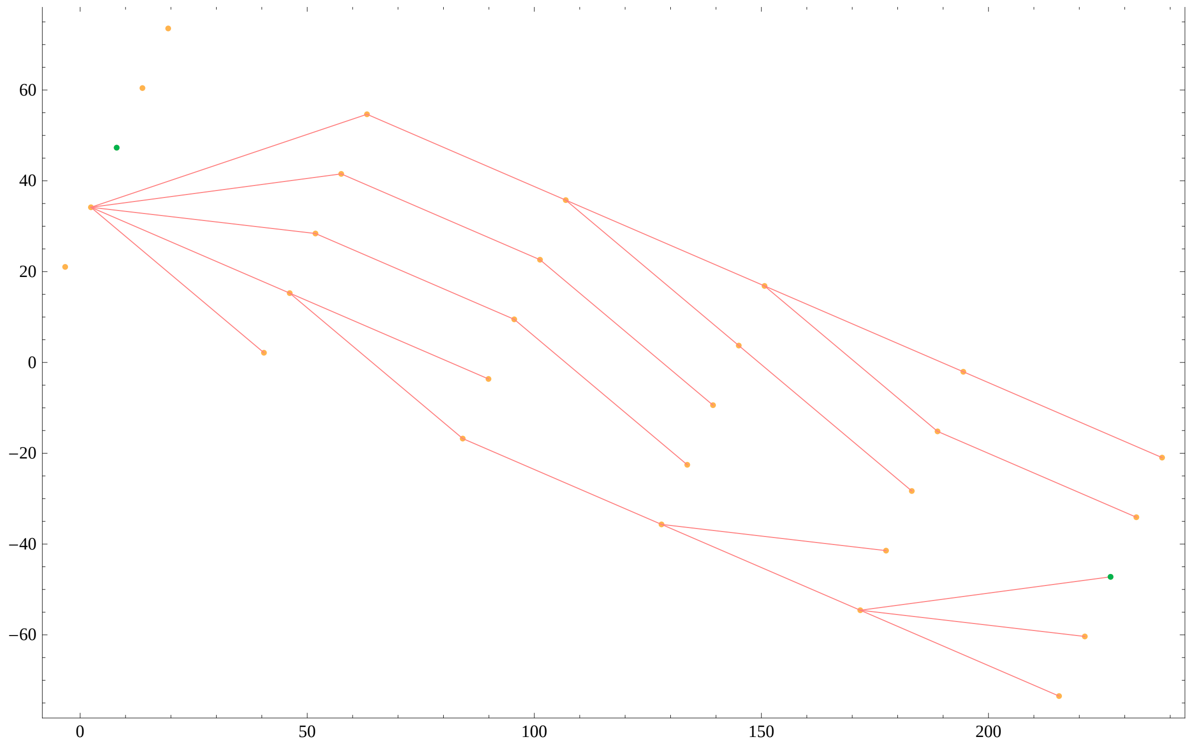
Finally, each element of the **Decision** matrix has the following structure:

$$\begin{pmatrix} \{\{3,4\},\{4,3\}\} \\ 8017.32 \end{pmatrix} \text{with} \begin{pmatrix} \{\{\text{previous node}\},\{\text{node}\}\} \\ \text{time of arrival} \end{pmatrix}.$$

```
In[245]:= Decision = DynProgr[2];
          Dtable = Decision // MatrixForm
Out[246]//MatrixForm=
 ( (    0    ) ( {{1, 2}, {2, 1}} ) ( {{2, 2}, {3, 1}} ) ( {{3, 1}, {4, 1}} ) ( {{4, 1}, {5, 1}} ) ( {{5, 1}, {6, 1}} ) )
 ( ( 1000000 ) (     2868.47      ) (      5578.4      ) (     8165.43      ) (     10288.8      ) (     11837.1      ) )
 (    ( 1 )    ( {{1, 2}, {2, 2}} ) ( {{2, 2}, {3, 2}} ) ( {{3, 3}, {4, 2}} ) ( {{4, 1}, {5, 2}} ) ( {{5, 1}, {6, 2}} ) )
 (    ( 0 )    (     2615.33      ) (     5451.38      ) (     8195.06      ) (      10479.      ) (      11881.      ) )
 ( (    0    ) ( {{1, 2}, {2, 3}} ) ( {{2, 3}, {3, 3}} ) ( {{3, 4}, {4, 3}} ) ( {{4, 4}, {5, 3}} ) ( {{5, 1}, {6, 3}} ) )
 ( ( 1000000 ) (     2534.11      ) (      5244.3      ) (     8017.32      ) (     10665.6      ) (     12363.8      ) )
 ( (    0    ) ( {{1, 2}, {2, 4}} ) ( {{2, 4}, {3, 4}} ) ( {{3, 5}, {4, 4}} ) ( {{4, 5}, {5, 4}} ) ( {{5, 4}, {6, 4}} ) )
 ( ( 1000000 ) (      2603.5      ) (     5066.86      ) (     7746.41      ) (     10443.2      ) (      13060.      ) )
 ( (    0    ) ( {{1, 2}, {2, 5}} ) ( {{2, 5}, {3, 5}} ) ( {{3, 5}, {4, 5}} ) ( {{4, 5}, {5, 5}} ) ( {{5, 5}, {6, 5}} ) )
 ( ( 1000000 ) (      2773.7      ) (     4894.73      ) (     7480.21      ) (      10308.      ) (     13105.5      ) )
```

```
In[247]:= G0 = Table[Graphics[{RGBColor[1, 0.7, 0.3], Point[Loc[[i, j]]]}], {i, 1, Maxi}, {j, 1, Maxj}];
          G1 = Table[Graphics[{RGBColor[1, 0.7, 0.3], Point[Loc[[i, j]]]}], {i, 1, Maxi}, {j, 1, Maxj}];
          G2 = Select[Flatten[Decision, 1], (Last[#] < 1 000 000) && (Last[#] > 0.1) &];
          DimG2 = First[Dimensions[G2]];
          G3 = Table[First[G2[[i]]], {i, 1, DimG2}];
          G4 = Table[{Loc[[First[G3[[i]]][[1]], First[G3[[i]]][[2]]]], Loc[[Last[G3[[i]]][[1]], Last[G3[[i]]][[2]]]]}, {i, 1, DimG2}];
          G5 = Table[Graphics[{RGBColor[1, 0.5, 0.5], Line[{G4[[i, 1]], G4[[i, 2]]}]}], {i, 1, DimG2}];
          Show[G1, G5, GD, GA, Frame → True]
```

Out[254]=



```
In[159]:= G6 = Table[{N[X[G4[[i, 1]]]], N[X[G4[[i, 2]]]]}, {i, 1, DimG2}];
          G7 = Table[Graphics3D[{RGBColor[1, 0.5, 1], Line[{G6[[i, 1]], G6[[i, 2]]}]}], {i, 1, DimG2}];
          Earth = Graphics3D[Sphere[{0, 0, 0}]];
          Show[Earth, G7]
```