Ajustement de faisceaux

Equation de projection

Equation "principale", de projection classique avec les notations ;

1. le point terrain P_j visible and k
2. le senseur k avec une fonction de projection terrain-image π_k;
3. le point image $q_{j,k}$ visible and k

Pour tous les point où P_j est visible dans π_k :

$$\pi_k(P_j) = q_{j,k} \tag{1}$$

Ici π_k est inconnu, $q_{j,k}$ est une observation et P_j peut être suivant les cas :

1. une observation ;
2. une inconnue avec des observations;
3. une inconnue sans obervation (point de liaison multiple ou non).

Formalisation

En plus de l'équation "principale" 1, on un certain nombre d'équation auxiliaires .(bloc, GPS ...) non détaillées ici A_i. Comme on a (beaucoup) plus d'inconnues que d'équations , on formalise le problème par moindre carrés :

$$\sum_{j,k} |\pi_k(P_j) - q_{j,k}|^2 + \sum_i A_i^2 \tag{2}$$

Comme le système n'est pas linéaire on le linéarise par différenciation. Comme la linérisation est une approximation on itère.

Implementation

Le diable est dans les détails . . . Il reste à définir notamment :

1. comment calculer les dérivées;
2. comment stocker les systèmes de moindres carrés;
3. comment gérer les grands systèmes avec des $10^4 \ldots 10^6$ points de liaisons;
4. comment résoudre les systèmes de moindres carrés;
5. quel modèles de caméra;

Calcul des dérivées-1

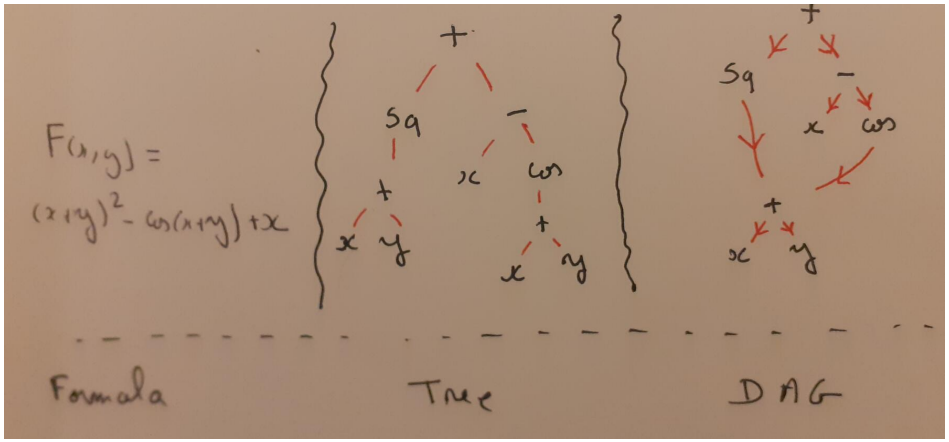
Les principales méthodes de calcul des dérivées sont :

1. à la main ; + efficace , − source d'erreur, difficile à maintenir;
2. dérivées numérique ; + simple, − lent, risque d'imprécision;
3. jet : + exact , − un peu lent , un peu compliqué à développer (NB un jet = 1 nombre + un infiniment petit formel, voir CERES)
4. dérivées symboliques : + exact et rapide , − moyennement compliqué à développer (mais assez simple à utiliser)
5. code differenciation : + exact et rapide, permet de dériver des expression avec des boucles − très compliqué à développer et assez compliqué à utiliser.

La méthode utilisée dans MicMac (V1 et V2) est celle des dérivées symboliques.

Calcul des dérivées-2

Représentation des formules comme un arbre :



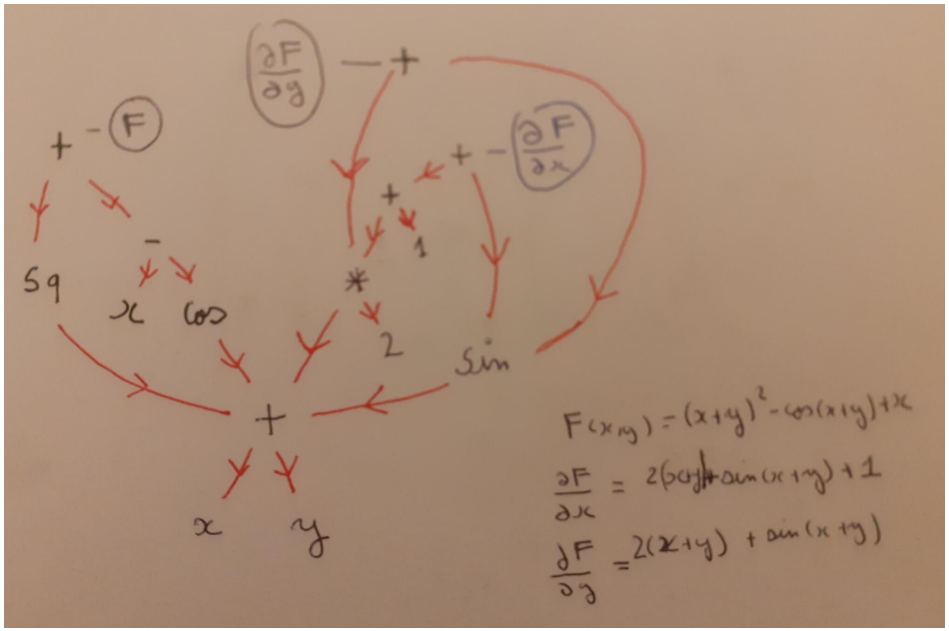
On définit toute les opérations sur les arbres de formules.

Calcul des dérivées-3

```
begin{lstlisting}[language=c++]
// === Extract of code in class cMulF (multiplication of formulas )

// method for computing values
void ComputeBuf(int aK0,int aK1) override
{
    for (int aK=aK0 ; aK<aK1 ; aK++)
        mDataBuf[aK] =  mDataF1[aK] * mDataF2[aK];
}

// method for computing the derivative
cFormula<TypeElem> Derivate(int aK) const override
{
   return  mF2*mF1->Derivate(aK) + mF1*mF2->Derivate(aK);
}

// method for generating the code
std::string GenCodeDef() const override
{
    return "(" + mF1->GenCodeRef() + "␣" + this->NameOperator() +  "␣" + mF2->GenCodeRef() +
}
```

Calcul des dérivées-4

Les formules identiques sont identifiées, le code est représenté comme un DAG :



Calcul des dérivées-5

Les principales règles de réductions sont utilisées :

- $0 * F \rightarrow 0$, $1 * F \rightarrow F$, $-1 * F \rightarrow -F$ (and idem $F * 0 \ldots$);
- $0/F \rightarrow 0$;
- $0 + F \rightarrow F$, (and idem $F + 0 \ldots$);
- $-(-F) \rightarrow F$, $F_1 - (-F_2) \rightarrow F_1 + F_2$;
- $F_1 * F_2 + F_1 * F_3 \rightarrow F_1 * (F_2 + F_3)$, $F_1 * F_2 - F_1 * F_3 \rightarrow F_1 * (F_2 - F_3)$,
- $F_2/F_1 + F_3/F_1 \rightarrow (F_2 + F_3)/F1$, $F_2/F_1 - F_3/F_1 \rightarrow (F_2 - F_3)/F1$,
- if $F_1 > F_2$ then $F_1 + F_2 \rightarrow F_2 + F_1$, here the comparison $F_1 > F_2$ is made on numbering, this rules is here favorize merging in DAG creation , any comparison as long as it anti-symetric would hold; we just want to avoid that in the same formula $F_1 + F_2$ and $F_2 + F_1$ cannot be merged;
- if $F_1 > F_2$ then $F_1 * F_2 \rightarrow F_2 * F_1$
- $C(a) \otimes C(b) \rightarrow C(a \otimes b)$ where $C(x)$ design the formula for constant x and $\otimes$ is any bynary operator;
- $\alpha(C(a)) \rightarrow C(\alpha(a))$ where α design any unary operator;

Calcul des dérivées-6

Le code généré ressemble à quelques chose comme cela :

```
double F29 = std::sin(F8);
double F13 = MMVII::DerSinC(F8);
double F14 = (2 * F8);
double F9 = MMVII::sinC(F8);
double F16 = (F15 / F14);
double F10 = (F9 * PixI);
double F22 = (F21 / F14);
double F11 = (F9 * PixJ);
double F18 = (F16 * F13);
double F32 = -(F22);
double F24 = (F22 * F13);
```

Modélisation des caméras, cas spatial-1

Pas encore fait dans V2, mais on compte reprendre en partie le même principe que dans V1:

1. hypothèse : la position est précise (GPS sans obstruction), l'imprécision vient de l'orientation;
2. la correction est une déformation image lisse, indépendante du relief;

Modèle de projection :

$$\pi = D \circ \pi_0 \tag{3}$$

On se donne pour D un modèle paramétrique de type polynomial en x, y.

Modélisation des caméras, cas spatial-1

Eventuellement on le contraint localement à être proche des rotations , avec O, P, K trois fonctions de x à variation lente.

$$D(x,y) \approx O(x)\frac{\partial \pi_0}{\partial \omega} + P(x)\frac{\partial \pi_0}{\partial \phi} + K(x)\frac{\partial \pi_0}{\partial \kappa} \tag{4}$$

En pratique dans la majorité des chantier traités en recherche (emprise limitée) on se limite au degré 1 en x, y et on n'utilise peu 4. Cette formalisation serait probablement à revoir dans le cadre d'un usage IGN sur de longues bandes.