# Network-based High Accuracy Positioning with the GPSTk

D. Salazar, M. Hernandez-Pajares, J.M. Juan-Zornoza, J. Sanz

Research group of Astronomy and Geomatics (gAGE)
Universitat Politecnica de Catalunya (UPC)

December 9th, 2010

# Outline

# Introduction

# Introduction

- This work presents a postprocess kinematic positioning technique.
- The technique is called *Precise Orbits Positioning* (POP).
- Kinematic PPP-like processing based on a network of stations.
- Satellite clock offsets will be estimated on-the-fly.
- It is independent of precise clocks, only needs precise orbits.
- Solution rate is only limited by data rate.
- PPP results rate limited by precise SV clocks data rate.

# Introduction

- We don't claim this strategy is original:
  - Network based clocks.
  - Phase interpolations.
- However:
  - *The implementation procedure is original*.
  - *Predicted and Rapid orbits may be used*.
  - *Broadcast orbits may yield acceptable results*.
- The technique is implemented using GPSTk tools.
- Reference implementation is provided as a GPSTk example.

# Introduction

- We don't claim this strategy is original:
  - Network based clocks.
  - Phase interpolations.
- However:
  - *The implementation procedure is original*.
  - *Predicted and Rapid orbits may be used*.
  - *Broadcast orbits may yield acceptable results*.
- The technique is implemented using GPSTk tools.
- Reference implementation is provided as a GPSTk example.

# Introduction

- We don't claim this strategy is original:
  - Network based clocks.
  - Phase interpolations.
- However:
  - *The implementation procedure is original*.
  - *Predicted and Rapid orbits may be used*.
  - *Broadcast orbits may yield acceptable results*.
- The technique is implemented using GPSTk tools.
- Reference implementation is provided as a GPSTk example.

# Introduction

- We don't claim this strategy is original:
  - Network based clocks.
  - Phase interpolations.
- However:
  - *The implementation procedure is original*.
  - *Predicted and Rapid orbits may be used*.
  - *Broadcast orbits may yield acceptable results*.
- The technique is implemented using GPSTk tools.
- Reference implementation is provided as a GPSTk example.

# POP description

1. Introduction
2. POP description
3. POP implementation
4. POP data processing
5. Conclusions

# POP description

## Master reference station

*Master* station clock is the reference for all the network

$$\delta Pc_0^j = M_0^j.ztd_0 - c.dt^j$$

$$\delta Lc_0^j = M_0^j.ztd_0 + bc_0^j - c.dt^j$$

Where:

- $\delta Pc_0^j$ and $\delta Lc_0^j$ : Prefilter residuals $SV^j$ and Master.
- $M_0^j$ : Tropospheric mapping function (Niell).
- $ztd_0$ : Zenith tropospheric path delay.
- $c.dt^j$ : Relative clock delay between $SV^j$ and Master.
- $bc_0^j$ : Ionosphere-free carrier phase ambiguity.

# POP description

## Master reference station

*Master* station equations set satellite clocks

$$\delta Pc_0^j = M_0^j.ztd_0 - c.dt^j$$

$$\delta Lc_0^j = M_0^j.ztd_0 + bc_0^j - c.dt^j$$

Where:

- $\delta Pc_0^j$ and $\delta Lc_0^j$ : Prefilter residuals $SV^j$ and Master.
- $M_0^j$ : Tropospheric mapping function (Niell).
- $ztd_0$ : Zenith tropospheric path delay.
- $c.dt^j$ : Relative clock delay between $SV^j$ and Master.
- $bc_0^j$ : Ionosphere-free carrier phase ambiguity.

**Reference** stations are similar to *Master*, but adding clock offsets

$$\delta Pc_k^j = M_k^j.ztd_k + c.dt_k - c.dt^j$$

$$\delta Lc_k^j = M_k^j.ztd_k + bc_k^j + c.dt_k - c.dt^j$$

Where:

- $c.dt_k$ : Relative clock delay between reference station $k$ and Master.

**Reference** station equations provide robustness

$$\delta Pc_k^j = M_k^j.ztd_k + c.dt_k - c.dt^j$$

$$\delta Lc_k^j = M_k^j.ztd_k + bc_k^j + c.dt_k - c.dt^j$$

Where:

- $c.dt_k$ : Relative clock delay between reference station $k$ and Master.

*Rover* has equations like PPP, but adding satellite clock offsets

$$
\begin{aligned}
\delta Pc_r^j &= \left(\frac{x_{r0}-x^j}{\rho_{r0}^j}\right)dx + \left(\frac{y_{r0}-y^j}{\rho_{r0}^j}\right)dy + \left(\frac{z_{r0}-z^j}{\rho_{r0}^j}\right)dz \\
&\quad + M_r^j.ztd_r + c.dt_r - c.dt^j \\
\delta Lc_r^j &= \left(\frac{x_{r0}-x^j}{\rho_{r0}^j}\right)dx + \left(\frac{y_{r0}-y^j}{\rho_{r0}^j}\right)dy + \left(\frac{z_{r0}-z^j}{\rho_{r0}^j}\right)dz \\
&\quad + M_r^j.ztd_r + bc_r^j + c.dt_r - c.dt^j
\end{aligned}
$$

- $(x_0, y_0, z_0)$ : A priori *rover* receiver position.
- $(x^j, y^j, z^j)$ : Satellite $SV^j$ position.
- $(dx, dy, dz)$ : Correction parameters to $(x_0, y_0, z_0)$.

# POP description

## Some additional remarks

- Ionosphere-free combination of observations are used.
- Connection between receivers is achieved by simultaneous estimation of SV clock offsets.
- Although the observations are not explicitly differentiated:
  - System of equations is equivalent to carrier phase-based differential DGPS.
  - Simultaneous estimation of satellite clock offsets allow them to become the *constraints* between equations.
- As said, POP allows rover precise positioning without precise satellite clock products.
- Other products (predicted/rapid/broadcast) may be used.
  - *This is an important advantage regarding other methods.*

# POP description

## Some additional remarks

- Ionosphere-free combination of observations are used.
- Connection between receivers is achieved by simultaneous estimation of SV clock offsets.
- Although the observations are not explicitly differentiated:
  - System of equations is equivalent to carrier phase-based differential DGPS.
  - Simultaneous estimation of satellite clock offsets allow them to become the *constraints* between equations.
- As said, POP allows rover precise positioning without precise satellite clock products.
- Other products (predicted/rapid/broadcast) may be used.
  - *This is an important advantage regarding other methods*.

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
  - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
  - some are *satellite-indexed* ($dt^j$),
  - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

- Implementation of an equation system like the former is complex.
- The system involves multiple stations separated hundreds of kilometers.
- There is a great number of unknowns of several kinds:
    - Some unknowns are *receiver-indexed* ($ztd_i$, $dx$, $dy$, $dz$, $c.dt_r$),
    - some are *satellite-indexed* ($dt^j$),
    - others are both *receiver-* and *satellite-indexed*, like $Bc_i^j$.
- Number of unknowns at given epoch has a wide variation (available station data, visible SV's).

# POP implementation

`SolverGeneral` class helps implementing this kind of systems

## Main idea behind `SolverGeneral`

- Equations and variables are *described*, not *hard-coded*.
- Programmer provides corresponding stochastic models and relationships.
- At each epoch the `SolverGeneral` object will:
  - Match incoming data (observations and ephemeris) with equations and variables descriptions.
  - Build and solve the appropriate equation system *for that epoch*.

# POP implementation

`SolverGeneral` class helps implementing this kind of systems

## Main idea behind `SolverGeneral`

- Equations and variables are *described*, not *hard-coded*.
- Programmer provides corresponding stochastic models and relationships.
- At each epoch the `SolverGeneral` object will:
  - Match incoming data (observations and ephemeris) with equations and variables descriptions.
  - Build and solve the appropriate equation system *for that epoch*.

# POP implementation

`SolverGeneral` class helps implementing this kind of systems

## Main idea behind `SolverGeneral`

- Equations and variables are *described*, not *hard-coded*.
- Programmer provides corresponding stochastic models and relationships.
- At each epoch the `SolverGeneral` object will:
  - Match incoming data (observations and ephemeris) with equations and variables descriptions.
  - Build and solve the appropriate equation system *for that epoch*.

# POP implementation

`SolverGeneral` class helps implementing this kind of systems

## Main idea behind `SolverGeneral`

- Equations and variables are *described*, not *hard-coded*.
- Programmer provides corresponding stochastic models and relationships.
- At each epoch the `SolverGeneral` object will:
  - Match incoming data (observations and ephemeris) with equations and variables descriptions.
  - Build and solve the appropriate equation system *for that epoch*.

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

First: Declaration and initialization of `Variable` objects

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

Stochastic models to be used

# POP implementation

## Example

```
WhiteNoiseModel coordinatesModel( 100.0 );
TropoRandomWalkModel tropoModel;
PhaseAmbiguityModel ambiModel;

Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

Variable cdt( TypeID::cdt );
   cdt.setDefaultForced(true);   // Force coefficient (1.0)

Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

Variable ambi( TypeID::BLC, &ambiModel, true, true );
   ambi.setDefaultForced(true);   // Force coefficient

Variable satClock( TypeID::dtSat, false, true );
   satClock.setDefaultCoefficient(-1.0); // Set coefficient
   satClock.setDefaultForced(true);      // Force coefficient

Variable prefitPC( TypeID::prefitC );
Variable prefitLC( TypeID::prefitL );
```

`Variables` for coordinates

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

`Variable` for rover clock offset

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

`Variable` for wet troposphere estimation

# POP implementation

## Example

```
WhiteNoiseModel coordinatesModel( 100.0 );
TropoRandomWalkModel tropoModel;
PhaseAmbiguityModel ambiModel;

Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

Variable cdt( TypeID::cdt );
   cdt.setDefaultForced(true);   // Force coefficient (1.0)

Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

Variable ambi( TypeID::BLC, &ambiModel, true, true );
   ambi.setDefaultForced(true);   // Force coefficient

Variable satClock( TypeID::dtSat, false, true );
   satClock.setDefaultCoefficient(-1.0); // Set coefficient
   satClock.setDefaultForced(true);      // Force coefficient

Variable prefitPC( TypeID::prefitC );
Variable prefitLC( TypeID::prefitL );
```

`Variable` describing phase ambiguities

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

`Variable` describing satellite clock offsets

# POP implementation

## Example

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set coefficient
       satClock.setDefaultForced(true);      // Force coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

“Dummy” `Variables` describing independent terms

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Second: Declaration of `Equation` objects

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Declare `Equation` for *Master* pseudorange

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Add `Variables` to *Master* pseudorange `Equation`

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Set what receiver (*Master*) this equation applies to

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Declare `Equation` for *Master* phase

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Add `Variables` to *Master* phase `Equation`

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Set what receiver (*Master*) this equation applies to

# POP implementation

## Example

```
Equation equPCMaster( prefitPC )

equPCMaster.addVariable( tropo );
equPCMaster.addVariable( satClock );
equPCMaster.header.equationSource = master;

Equation equLCMaster( prefitLC );

equLCMaster.addVariable( tropo );
equLCMaster.addVariable( satClock );
equLCMaster.addVariable( ambi );
equLCMaster.header.equationSource = master;

equLCMaster.setWeight( 10000.0 );
```

Set the *relative weight* of *Master* phase `Equation`

## Example

```
EquationSystem equSystem;

equSystem.addEquation( equPCRover );
equSystem.addEquation( equLCRover );
equSystem.addEquation( equPCRef );
equSystem.addEquation( equLCRef );
equSystem.addEquation( equPCMaster );
equSystem.addEquation( equLCMaster );

SolverGeneral solver( equSystem );
```

Third: Declaration of `EquationSystem` and `SolverGeneral`

# POP implementation

## Example

```
EquationSystem equSystem;

equSystem.addEquation( equPCRover );
equSystem.addEquation( equLCRover );
equSystem.addEquation( equPCRef );
equSystem.addEquation( equLCRef );
equSystem.addEquation( equPCMaster );
equSystem.addEquation( equLCMaster );

SolverGeneral solver( equSystem );
```

Declare an `EquationSystem` object

# POP implementation

## Example

```
EquationSystem equSystem;

equSystem.addEquation( equPCRover );
equSystem.addEquation( equLCRover );
equSystem.addEquation( equPCRef );
equSystem.addEquation( equLCRef );
equSystem.addEquation( equPCMaster );
equSystem.addEquation( equLCMaster );

SolverGeneral solver( equSystem );
```

Add `Equations` to `EquationSystem` object

# POP implementation

## Example

```
EquationSystem equSystem;

equSystem.addEquation( equPCRover );
equSystem.addEquation( equLCRover );
equSystem.addEquation( equPCRef );
equSystem.addEquation( equLCRef );
equSystem.addEquation( equPCMaster );
equSystem.addEquation( equLCMaster );

SolverGeneral solver( equSystem );
```

Declare a `SolverGeneral`, feed it with `EquationSystem`

# POP data processing

# POP data processing

- This is a multi-station problem.
- Preprocess all stations, one by one, in a way similar to PPP.
- Results from preprocessing are stored in an appropriate multi-epoch, multi-station GNSS Data Structure (GDS).
- Extract one epoch of data each time from GDS, and feed solver.
- 5 IGS stations were used:
  - `ACOR`, `MADR`, `SCOA`, `SFER` and `TLSE`.
  - Network across Iberian Peninsula spanning 1023 km (`SFER-TLSE`).
  - Station `ACOR` was *Master*, and `MADR` was *Rover*.
  - `MADR` is 392 km away from nearest reference station (`SCOA`).

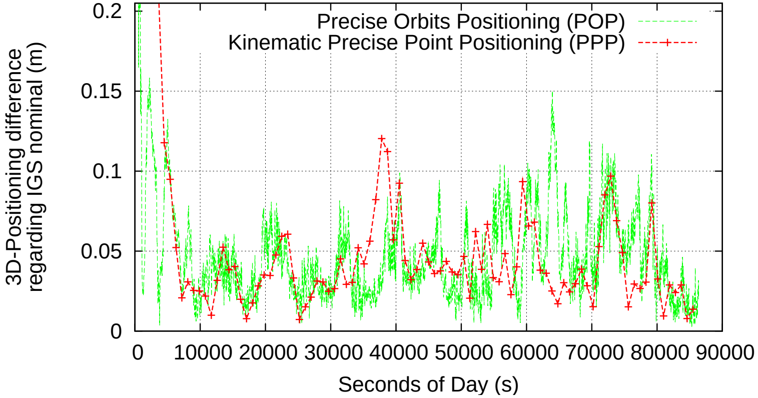
# POP data processing: POP test network


POP network. Rover: MADR (2008/05/27)

# POP data processing: POP test network

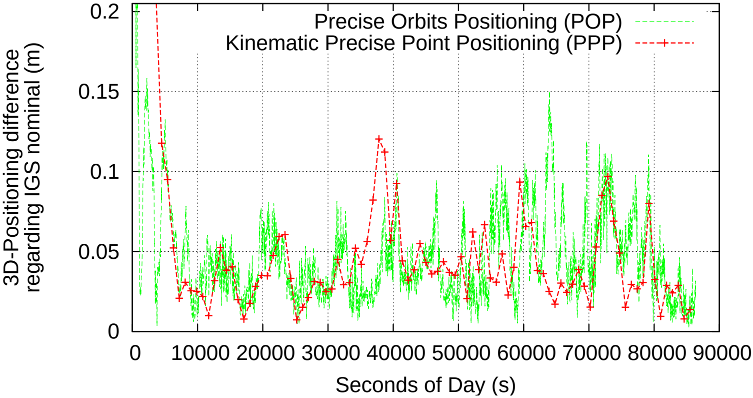

This network comprises more than 580,000 $km^2$

# POP data processing: MADR, 5-station network



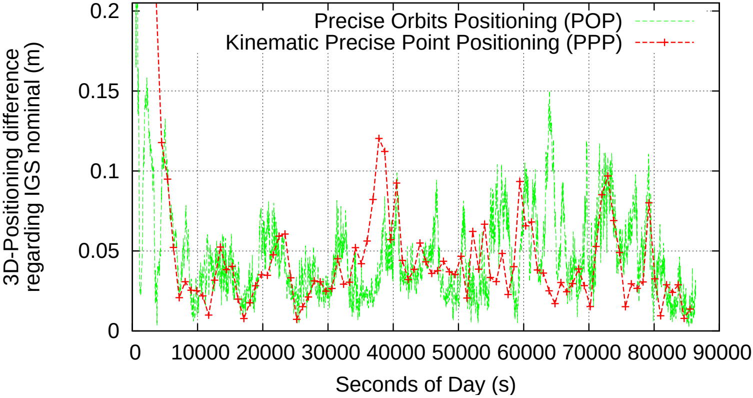

POP versus kinematic PPP processing. MADR 2008/05/27

# POP data processing: MADR, 5-station network



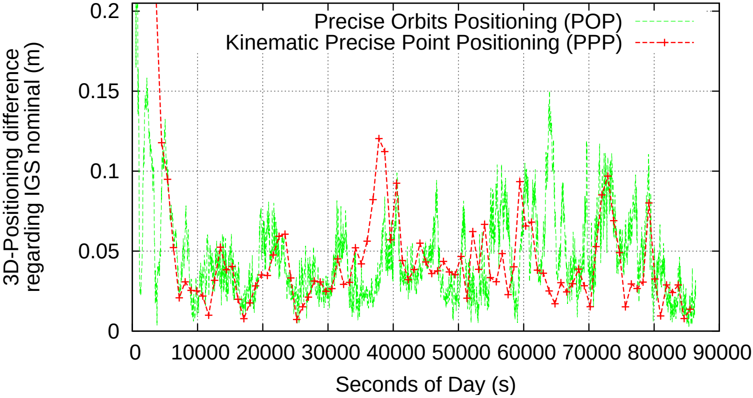

Results are very similar, as expected

# POP data processing: MADR, 5-station network




3D-RMS: 0.046 m for kinematic PPP vs. 0.049 m for POP

# POP data processing: MADR, 5-station network

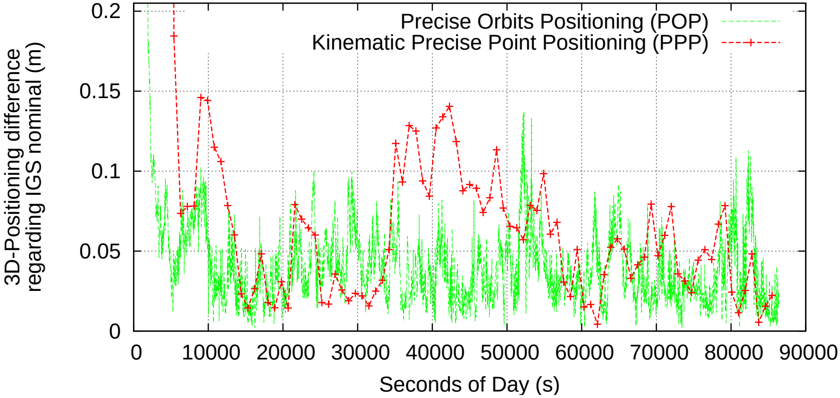



POP yields higher positioning rate

# POP data processing: POP test network

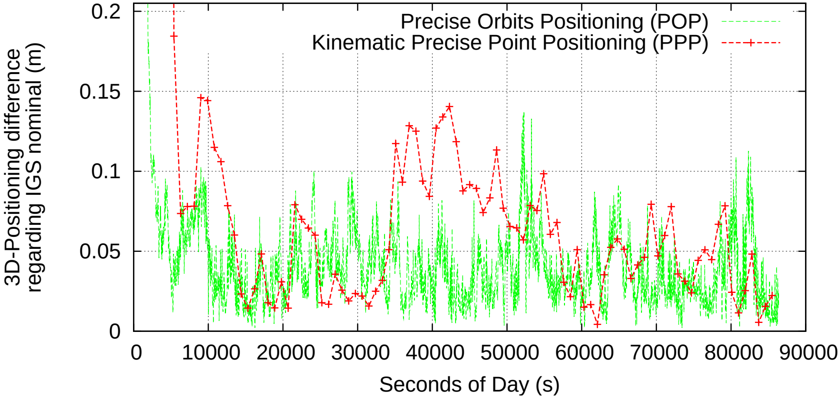

POP test network. Rover: TLSE (2008/05/27)

# POP data processing: TLSE, 5-station network



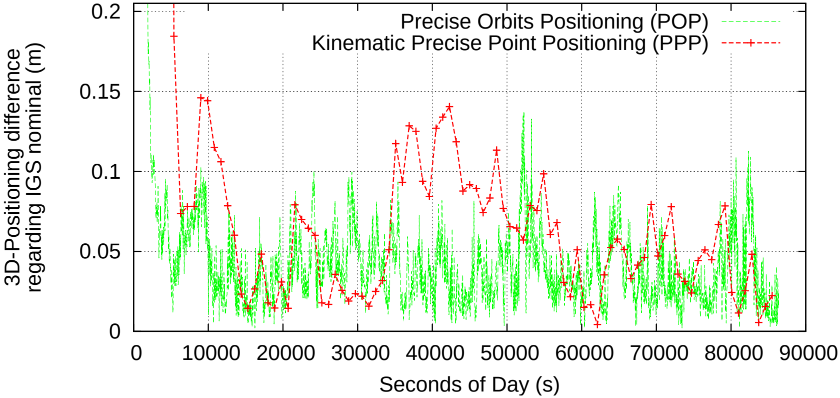

POP versus kinematic PPP processing. TLSE 2008/05/27

# POP data processing: TLSE, 5-station network



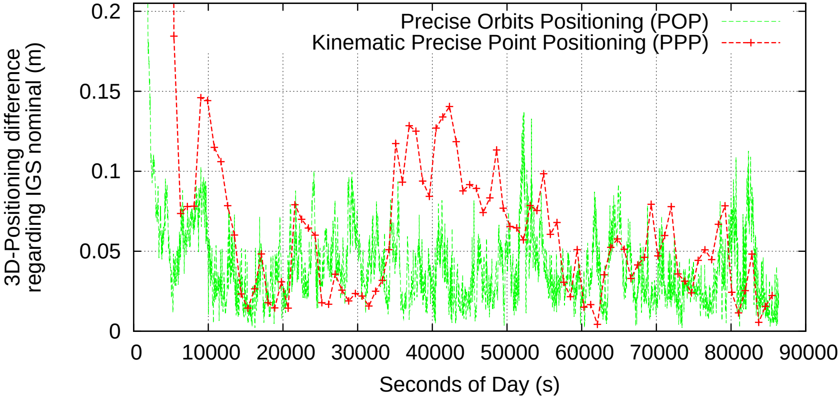

Rover (`TLSE`) is outside network, 257 km from nearest reference

# POP data processing: TLSE, 5-station network



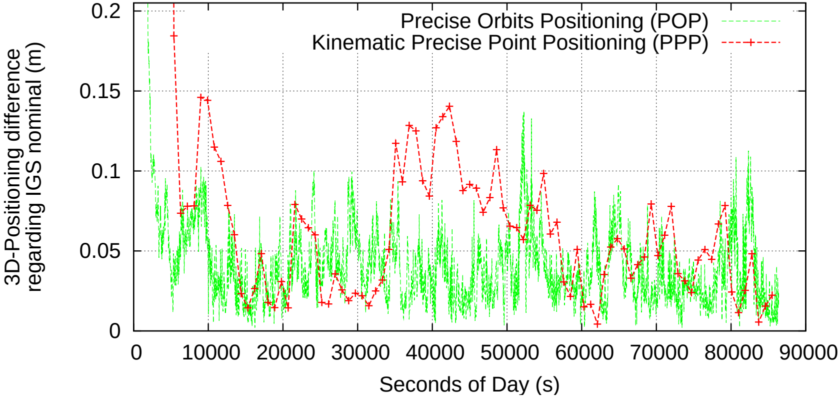

Network-base processing provides additional robustness

# POP data processing: TLSE, 5-station network




POP better between 35000-50000 s, kinematic PPP has problems

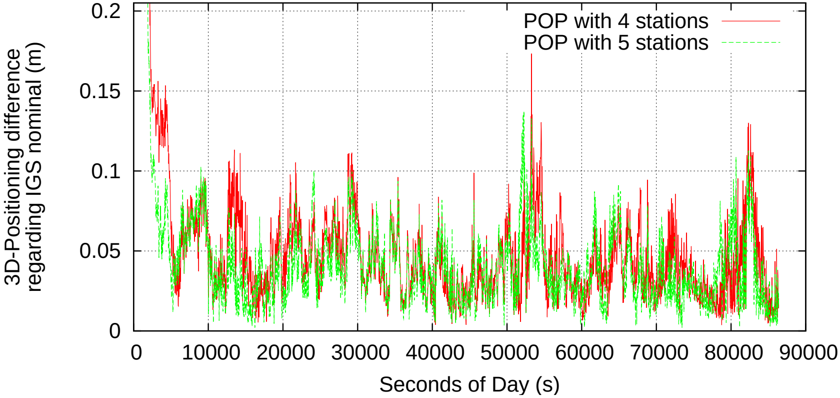
# POP data processing: TLSE, 5-station network




3D-RMS: 0.069 m for kinematic PPP vs. 0.044 m for POP

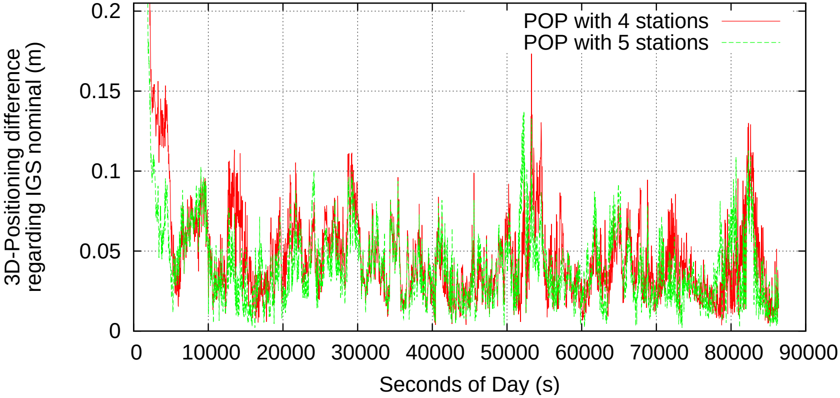

POP test network. Rover: TLSE, SCOA is deleted

# POP data processing: TLSE, 4-, 5-station network



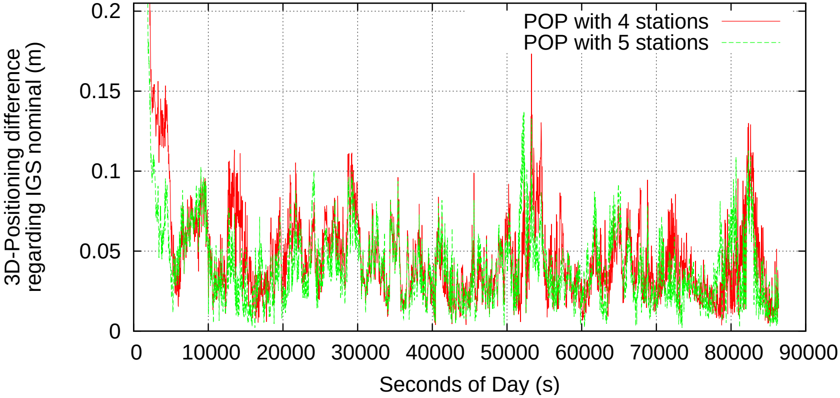

POP results for 4 and 5-stations networks. TLSE 2008/05/27

# POP data processing: TLSE, 4-, 5-station network



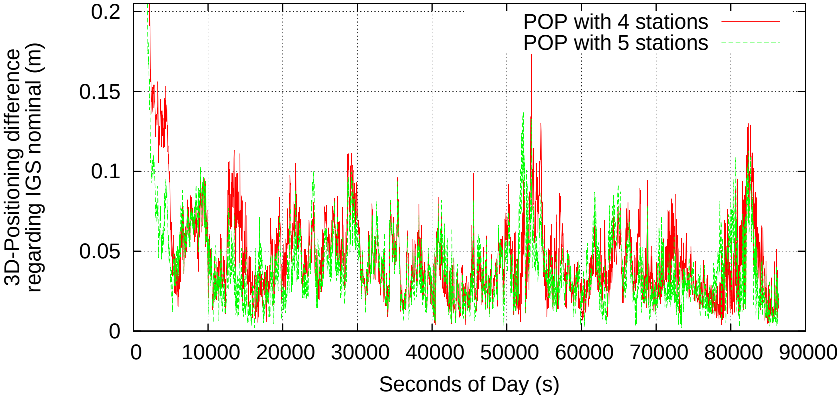

Distance from Rover to nearest reference station is not critical

# POP data processing: TLSE, 4-, 5-station network



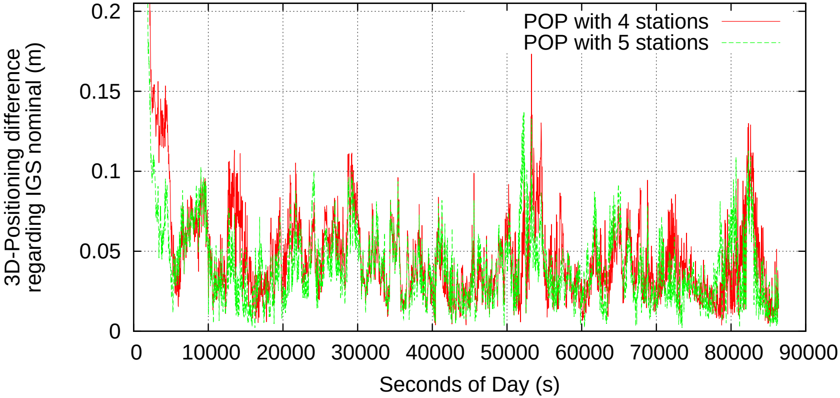

SCOA is taken out network leaving 4 stations (including Rover)

# POP data processing: TLSE, 4-, 5-station network




MADR is nearest reference station (588 km away)

# POP data processing: TLSE, 4-, 5-station network

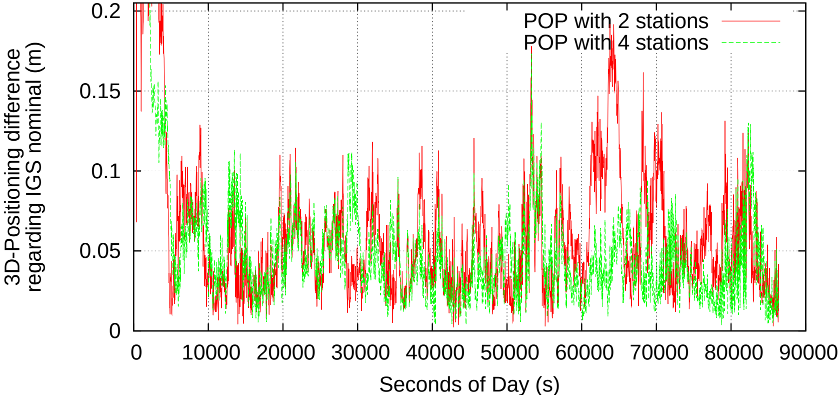



3D-RMS barely increases from 0.044 m to 0.049 m

# POP data processing: POP test network

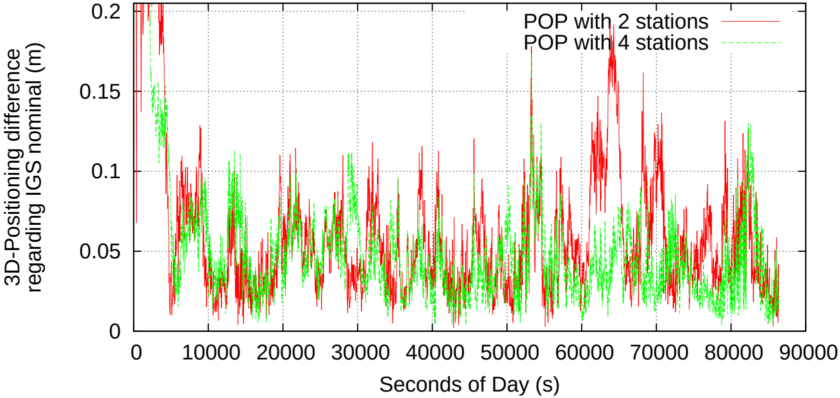

POP test network. Rover: TLSE, Master: MADR

# POP data processing: TLSE, 2-, 4-station network



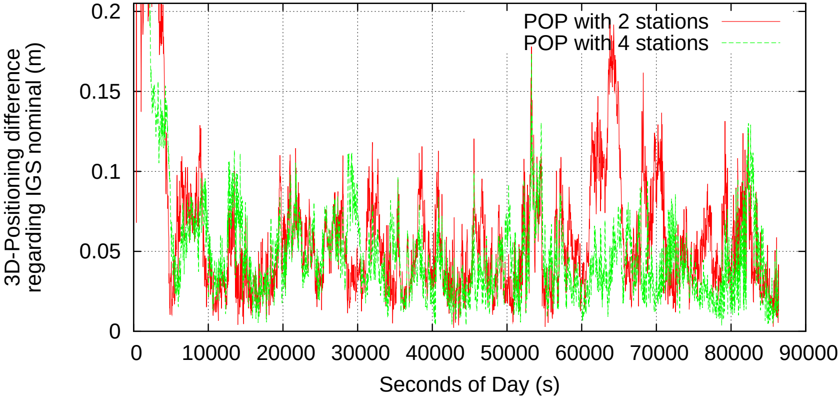

POP results for 2 and 4-stations networks. TLSE 2008/05/27

# POP data processing: TLSE, 2-, 4-station network



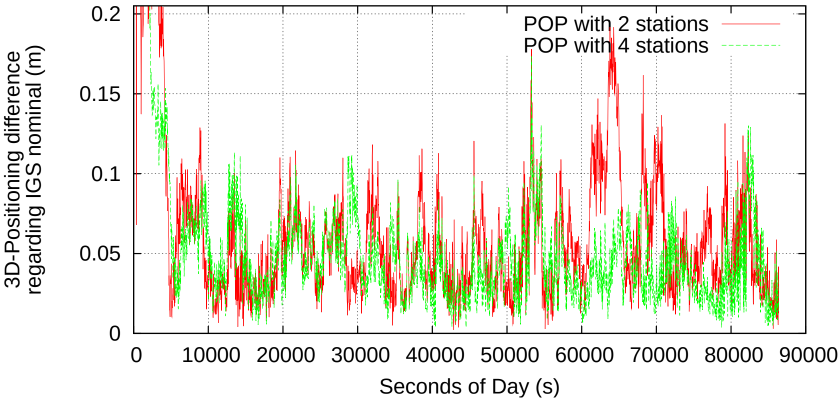

POP with only two stations becomes carrier phase-based DGPS

# POP data processing: TLSE, 2-, 4-station network



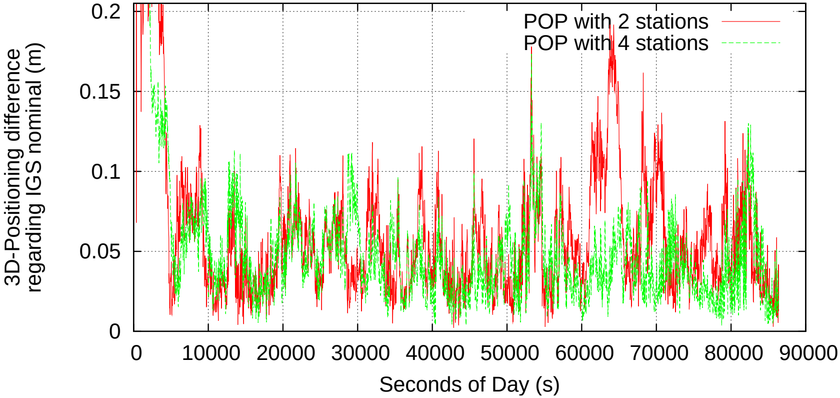

`MADR` as *Master* and `TLSE` as *Rover*: 588 km-long baseline

# POP data processing: TLSE, 2-, 4-station network



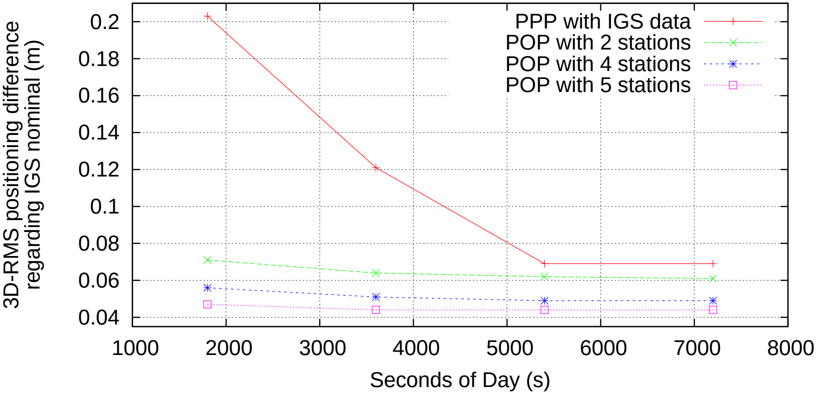

Probably not enough common satellites

# POP data processing: TLSE, 2-, 4-station network



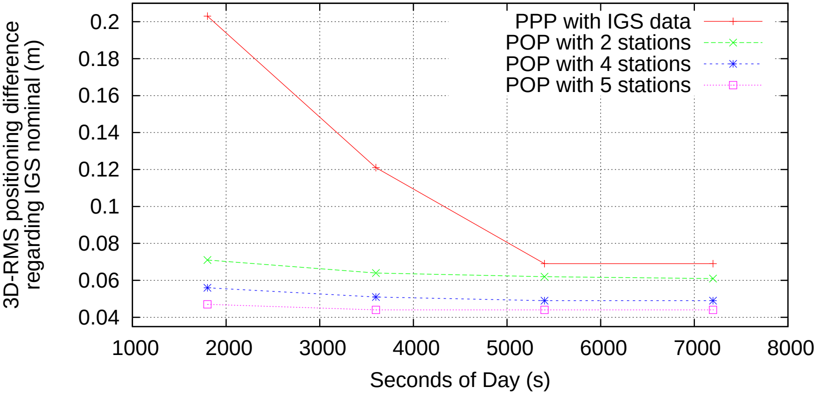

Degraded satellite clocks estimations

# POP data processing



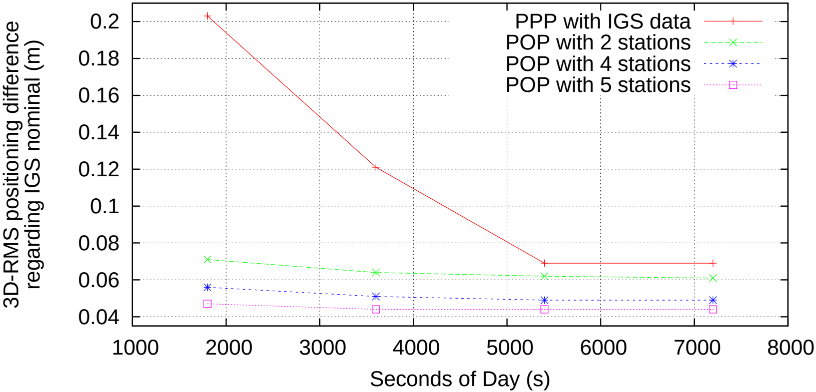

3D-RMS of error computed **from** 1800 s, 3600 s, 5400 s, 7200 s on

# POP data processing




3D-RMS of error improves as station number increases, up to a limit

# POP data processing




Convergence time is an issue

# Conclusions

# Conclusions

- Precise Orbits Positioning (POP) was implemented:
  - Uses a network of reference stations.
  - Independent of precise clock information.
  - Only needs orbits information to work at arbitrary data rates.
  - Allows precise positioning hundreds of kilometers from reference.
  - Provides additional robustness and flexibility.
  - Implemented in a novel way using `SolverGeneral`.
  - An open-source reference implementation is provided.

# Thanks for your attention!!!