PTGaussian
—
A Matlab class for working with (non-)linear covariance propagation for Multivariate Normally Distributed variables
or
Uncertainty propagation in Matlab without the Agonizing Pain
(propagation des incertitudes dans matlab sans la douleur atroce)

Niclas Börlin
niclas.borlin@cs.umu.se
Department of Computing Science
Umeå University
Sweden

December 1, 2010

Background

- Statistical error propagation and reasoning central to Photogrammetry.
- Used seldom in Computer Vision and Non-linear Optimization.
- Why?

Potential reasons for ignoring statistical properties:

- Difficult
  - Sometimes hard to estimate statistical properties for the measurements.
  - Error propagation expressions are complex in general.
  - At best, "simple" expressions exists only for the Gaussian distribution.

\[ C_{ab} = C_{bb} - A C_{xx} A^T, \]
\[ C_{ab} = M^{-1} - M^{-1} H (H^T M^{-1} H)^{-1} H^T M^{-1}, \]
\[ M = A C_{xx}^{-1} A, \]

- Expensive
  - Additional storage of \( O(n^2) \) (complete covariance matrix) for a \( n \)-vector.
  - Additional computation \( O(n^3) \) to calculate covariance propagation.
Introduction

Goal

- Make statistics more accessible, especially working with vectors of normally distributed variables.
- Construct a class in Matlab that
  - encapsulates vectors of multivariate normally distributed variables and their associated covariance matrices,
  - has functions and overloaded operators for
    - linear transformations,
    - estimation, and
    - testing,
  - emulates matrices and matrix-vector operations,
  - support graphical and numerical display of important statistical properties, and
  - has overloaded operators for non-linear transformations with first-order propagation and/or numerical re-sampling.
- Part of a larger effort: *The Photogrammetric Toolbox* in Matlab.

Applications

- Camera calibration
- 3D reconstruction
- “Photomodeler” and more

The Photogrammetric Toolbox in Matlab

Goal

- Implement all major photogrammetric algorithms, e.g. feature extraction, feature matching, resection, intersection, bundle adjustment, etc.
- Make it freely available in Matlab (Octave, C++, ...).
- Work with Euclidean and projective geometry.
- Work with 2D and 3D data.
- The toolbox will be object-oriented and have classes, e.g.
  - PTCamera, PTCamStation, ...
  - PTPoint2D, PTPoint3D, ...
  - etc.
  - PTGaussian

PTGaussian key points

- Visualization.
- Transformations.
- Estimations.
- Testing.
PTGaussian intended use

- Understand statistics and error propagation.
- Use in teaching.
- Use in research.

Example (Transformation: The camera equation)

The camera equation describes the projection of a 3D point 
\( \mathbf{X} = (X, Y, Z, 1)^T \) to a 2D point \( \mathbf{x} = (x, y, w)^T \) as

\[
\mathbf{K} \mathbf{R} (I - \mathbf{C}) \mathbf{X} = \mathbf{x},
\]

where

\[
\mathbf{K} = \begin{pmatrix}
  f & 0 & p_x \\
  0 & f & p_y \\
  0 & 0 & 1
\end{pmatrix}
\]

describes the internal camera parameters, \( \mathbf{R} \) is a 3D rotation matrix that describes the camera orientation in space, and \( \mathbf{C} \) is the camera position in space.

How does the uncertainties in the parameters in \( \mathbf{K}, \mathbf{R}, \mathbf{C}, \mathbf{X} \) propagate to \( \mathbf{x} \)?

Example (Transformation: The camera equation)

The camera equation

\[
\mathbf{K} \mathbf{R} (I - \mathbf{C}) \mathbf{X} = \mathbf{x},
\]

is equivalent to the collinearity equations

\[
x = x_p + f \frac{r_{11}(X - X_0) + r_{12}(Y - Y_0) + r_{13}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)},
\]

\[
y = y_p + f \frac{r_{21}(X - X_0) + r_{22}(Y - Y_0) + r_{23}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)}.
\]
Example (Estimation: Forward intersection)

- Given two observations of the same 3D point we have
  \[ K_1 R_1 (I - C_1) X = x_1, \]
  \[ K_2 R_2 (I - C_2) X = x_2. \]

- How does the uncertainties in the parameters in \( K_i, R_i, C_i \) and the observations \( x_i \) propagate to the estimated 3D point \( \hat{X} \)?

Classes

- PTGaussian — main Matlab class.
  - Emulates an \( m \)-vector of observations and their associated \( m \times m \) covariance matrix.
  - Can also emulate an \( mn \)-vector of observations stacked as an \( m \times n \) matrix.
- PTCoGaussian — helper class to enable encapsulation.

Density functions

- A cumulative density function (cdf) \( P_x(x) \) of a continuous stochastic variable \( x \) is defined such that
  \[ P_x(x) = P(x < x), \]
  i.e. \( P_x(x) \) is a non-decreasing function with
  \[ P_x(-\infty) = 0, \quad P_x(\infty) = 1. \]
- The corresponding probability density function (pdf) \( p_x(x) \) is defined as
  \[ p_x(x) = \frac{\partial P_x(x)}{\partial x} \in [0, 1] \]
  and thus
  \[ P_x(x) = \int_{-\infty}^{x} p_x(t) dt. \]

Thus, e.g.
\[ P(a < x \leq b) = P_x(b) - P_x(a), \quad P(x \geq c) = 1 - P_x(c). \]
**PTGaussian — A Matlab class for Multivariate Normally Distributed variables**

**Statistics**

**Basic statistics**

**Mean and variance**

- The *mean* $\mu_x$, *standard deviation* $\sigma_x$, and *variance* $\sigma_x^2$ of a stochastic variable $x$ are defined as
  
  \[
  \mu_x = E(x) = \int t p_x(t) dt,
  \]

  \[
  \sigma_x^2 = V(x) = E((x - \mu_x)^2),
  \]

  where $E(\cdot)$ is the *expectation operator*.

**Covariance**

- The *covariance* $\sigma_{xy}$ between two stochastic variables $x$ and $y$ is defined as
  
  \[
  \sigma_{xy} = \text{Cov}(x, y) = E((x - \mu_x)(y - \mu_y)),
  \]

- Observe that $\sigma_{xx} = \sigma_x^2$.

**Correlation**

- If the covariance is normalized by the standard deviations, we get the *correlation coefficient* $\rho_{xy}$, defined as
  
  \[
  \rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \in [-1, +1].
  \]

- If $\rho_{xy} = 0$, the variables $x$ and $y$ are said to be *uncorrelated*.
- Independent variables are uncorrelated.
- However, uncorrelated variables may still be dependent.
- High $|\rho_{xy}|$ values imply that the variables $x$ and $y$ are (almost) linearly dependent, i.e. cannot be estimated independently.
Mean and covariance for vectors

- For a stochastic $m$-vector $\mathbf{x}$, the mean vector $\mu_x$ is given by
  \[ \mu_x = E(\mathbf{x}) = \int t p_x(t) dt. \]

- The covariance matrix $C_{xx}$ within $\mathbf{x}$ is given by the $m \times m$ symmetric matrix
  \[ C_{xx} = \text{Cov}(\mathbf{x}, \mathbf{x}) = E \left( (\mathbf{x} - E(\mathbf{x})) (\mathbf{x} - E(\mathbf{x}))^T \right). \]

- If $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$, then $C_{xx} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_y^2 \end{pmatrix}$.

The Gaussian distribution

- A stochastic variable $\mathbf{x}$ with a pdf of
  \[ \phi_x(x) = \frac{1}{\sqrt{2 \pi} \sigma_x} e^{-\frac{1}{2} \left( \frac{x - \mu_x}{\sigma_x} \right)^2}, \sigma_x > 0, \]
  is called Normally or Gaussian distributed.

- Such a stochastic variable is denoted
  \[ \mathbf{x} \sim N(\mu_x, \sigma_x^2), \]
  where $\mu_x$ is the mean and $\sigma_x^2$ is the variance.

- The cdf $\Phi_x(x)$ is defined via integration
  \[ \Phi_x(x) = \int_{-\infty}^{x} \phi_x(t) dt. \]

Standardized Gaussian variables

- A standardized normal variable $\mathbf{y}$
  \[ \mathbf{y} = \frac{\mathbf{x} - \mu_x}{\sigma_x} \sim N(0,1) \]
  has $\mu_y = 0$ and $\sigma_y = 1$.

- Most of the probability is concentrated around $\mu_y = 0$.
  \[ P(|\mathbf{y}| \leq 1) = P(|\mathbf{x} - \mu_x| \leq 1\sigma_x) \approx 0.6827 \]
  \[ P(|\mathbf{y}| \leq 2) = P(|\mathbf{x} - \mu_x| \leq 2\sigma_x) \approx 0.9545 \]
  \[ P(|\mathbf{y}| \leq 3) = P(|\mathbf{x} - \mu_x| \leq 3\sigma_x) \approx 0.9973 \]
The $\alpha$ confidence interval for a Gaussian variable $x$ is given by $\mu_x \pm k_\alpha \sigma_x$, where the quantile constant $k_\alpha = \Phi^{-1}(1 - (1 - \alpha)/2)$, is defined such that $P(|x - \mu_x| \leq k_\alpha \sigma_x) = \alpha$.

If e.g. $\alpha = 0.95$, $k_{0.95} = \Phi^{-1}(0.975) \approx 1.96$.

With the spectral decomposition $C = V \Lambda V^T = (v_1 \quad v_2) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} (v_1 \quad v_2)^T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^T$, we find the eigenvalues $\lambda_1 = \sigma_x^2$, $\lambda_2 = \sigma_y^2$ and eigenvectors $v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

If the two stochastic variables $x \sim N(\mu_x, \sigma_x^2)$, $y \sim N(\mu_y, \sigma_y^2)$ are independent, their joint density function is $p_{xy}(x, y) = \frac{1}{2\pi \sigma_x \sigma_y} e^{-\frac{1}{2} \left( \frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} \right)}$.

With the vectors and $2 \times 2$ matrix $z = \begin{pmatrix} x \\ y \end{pmatrix}$, $\mu_z = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$, $C = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$, this can be written as $p_z(z) = \frac{1}{2\pi \sqrt{|C|}} e^{-\frac{1}{2} (z-\mu)^T C^{-1} (z-\mu)}$.

The 2D normal distribution is an elliptic bell-shaped function and may be defined by its elliptic contour lines defined by $(z-\mu)^T C^{-1} (z-\mu) = k$. 

![Diagram of a 2D normal distribution with contour lines](image-url)
For a general, positive semidefinite matrix, we have
\[ C = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} = (v_1 \quad v_2) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} (v_1 \quad v_2)^T. \]

The quantile constant \( \chi^2_{r, \alpha} \) is defined such that
\[ P(z < \chi^2_{r, \alpha}) = \alpha. \]

If \( \Phi_r(x) \) is the cdf of a \( \chi^2 \)-distribution with \( r \) degrees of freedom, then
\[ \chi^2_{r, \alpha} = \Phi_r^{-1}(\alpha). \]

\( \chi^2_{r, \alpha} \) is also known as the (upper) \( \alpha \) confidence limit for \( z \).

If \( y_i \sim N(0, 1) \) are standardized normal variables, and
\[ z = \sum_{i=1}^r y_i^2 \]

is the sum of \( r \) linearly independent random variables, then \( z \)
is chi-square distributed with \( r \) degrees of freedom
\[ z \sim \chi^2_r. \]

If \( z \in \mathbb{R}^2 \sim N(\mu_z, C_{zz}) \), the 95\% confidence intervals for the components are given by
\[ x = \mu_x \pm k_{0.95} \sigma_x = \mu_x \pm \sqrt{\chi^2_{1, 0.95} \sigma_x} \approx \mu_x \pm 1.96 \sigma_x, \]
\[ y = \mu_y \pm k_{0.95} \sigma_y = \mu_y \pm \sqrt{\chi^2_{1, 0.95} \sigma_y} \approx \mu_y \pm 1.96 \sigma_y. \]

The joint confidence limit for \( z \) is given by the ellipse satisfying
\[ (z - \mu_z)^T C_{zz}^{-1} (z - \mu_z) = \chi^2_{2, 0.95} = 5.99. \]
Hypothesis testing

- Given, \( x \sim N(\mu_x, C_{xx}) \), \( y \sim N(\mu_y, C_{yy}) \), and \( z = x - y \sim N(\mu_z, C_{zz}) \), we have
  \[
  P(x = y) = P(z = 0).
  \]

- Thus, the equality hypothesis
  \[
  H_0 : x = y
  \]
  can be replaced by
  \[
  H_0 : z = 0.
  \]

- If \( C_{zz} \) has full rank \( n \),
  \[
  z^T C_{zz}^{-1} z \sim \chi_n^2
  \]
  then \( H_0 \) may be rejected on the \( \alpha \) level if
  \[
  z^T C_{zz}^{-1} z > \chi_{n, \alpha}^2.
  \]
PTGaussian — A Matlab class for Multivariate Normally Distributed variables

The PTGaussian class

Construction

Construction

>> v=[3;4]; C=[3,2.5; 2.5,6];
>> p=PTGaussian(v,C) % General case, full covariance matrix.
   p = PTGaussian (2,1)
   mean = 3  std = 1.732  corr = +++++
   4  2.449  + + ++
>> w=[6;5]; D=diag([2,3]);
>> q=PTGaussian(w,D) % Uncorrelated elements.
   q = PTGaussian (2,1)
   mean = 3  std = 1.414  corr = +++++
   5  1.732  +++
>> r=PTGaussian(v) % Exact, covariance==0.
   r = PTGaussian (2,1) (exact)
   mean = 3  
   4  
>> s=PTGaussian([4;3],ones(2)) % Very dependent.
   s = PTGaussian (2,1)
   mean = std = corr =
   4+ 1 +++++  ++++
   3+ 1 +++++  ++++

Warning!

As default, no check is performed to ensure that the covariance matrix is symmetric positive semidefinite.

>> t=PTGaussian([1;2],[0,2;2,0])
   s = PTGaussian (2,1)
   mean = std = corr =
   1 0 +
   2 0 +

Use 'testcov' argument to warn if covariance matrix is incorrect or 'verifycov' to trip on it.

Internal storage

The covariance matrix is stored internally either as sparse (default) or full.

- Reduces memory use.
- May improve speed.
- Generally transparent to the user.
- May be overridden.

Command line display

Command line display

>> disp(p,0) % Default.
   PTGaussian (2,1)
   mean =
   3
   4
>> disp(p,1)
   PTGaussian (2,1)
   mean = std =
   3 1.732
   4 2.449
>> disp(p,2) % Default for non-exact column vectors.
   PTGaussian (2,1)
   mean = std = corr =
   3 1.732 100 59
   4 2.449 59 100
>> disp(p,3)
   PTGaussian (2,1)
   mean = std = corr =
   3 1.732 100 59
   4 2.449 59 100
>> disp(p,4) % "Raw" mode.
   PTGaussian (2,1)
   mean = cov =
   3 3 2.5
   4 2.5 6
**Tagging**

- **Exact elements are tagged with a '*'**.

```matlab
>> r
r =
PTGaussian (2,1) (exact)
mean =
    3*
    4*
```

- **Elements with high correlations are tagged with a '+'**.

```matlab
>> s
s =
PTGaussian (2,1)
mean =
    std =
    corr =
    5+ 1
    3+ 1
```

**Data access**

- **Object data are accessed through .-notation.**

```matlab
>> p.mean
ans =
    3
    4
>> p.cov
ans =
    (1,1) 3.0000
    (2,1) 2.5000
    (1,2) 2.5000
    (2,2) 6.0000
>> p.std
ans =
    1.7321
    2.4495
>> p.corr
ans =
    (1,1) 1.0000
    (2,1) 0.5893
    (1,2) 0.5893
    (2,2) 1.0000
```

- **p.var and p.std return the same size as p.mean.**
- **p.cov and p.corr return square matrices.**

**Plotting**

- **The bell curve for 1D data can be plotted.**

```matlab
>> plot(p(1),'b-') % Plot the bell curves for p(1) and q(1)
>> hold on, plot(q(1),'r--'), hold off
```

- **For clarity, the ‘full’ parameter is often used but not displayed in this presentation.**
The `PTGaussian` class

Data access

- Confidence ellipses(-oids) for 2D/3D data can be visualized.
  ```matlab
  >> plot(p,'b-') % Plot the 1-sigma ellipse.
  >> hold on, plot(q,'r--'), plot(r,'kx'), plot(s,'g.-'), hold off
  >> axis equal
  ```

- The `plot` function uses the `visconfreg` function to generate data for visualization of the confidence regions.
  ```matlab
  >> [X,Y,Z]=visconfreg(P,2); surf(X,Y,Z); % 2-sigma ellipsoid
  >> [x,y]=visconfreg(p,3); plot(x,y); % 3-sigma ellipse
  >> visconfreg(p(1),1.96) % 1.96-sigma confidence interval
  ans =
  -0.3948  6.3948
  >> visconfreg(p(2))
  ans =
  -0.8010  8.8010
  ```

Sampling

- Each `PTGaussian` corresponds to a distribution that can be sampled.
  ```matlab
  >> x=sample(p,100);
  >> plot(p,sqrt(5.99)) % 95% chi-square upper confidence limit
  >> axis equal, grid on
  >> set(gca,'xtick',visconfreg(p(1),1.96)) % Tick marks at 1.96s
  >> set(gca,'ytick',visconfreg(p(2),1.96))
  >> line(x(1,:),x(2,:),'marker','.','linestyle','none')
  ```
PTGaussian arrays

- Arrays can have any size, not just column vectors.
- The covariance matrix for an \( m \times n \) array has size \( mn \times mn \).

```matlab
>> M=[1,2,3;4,5,6]; C=diag(11:16).^2;
>> A=PTGaussian(M,C)
PTGaussian (2,3)
mean =
1 2 3
4 5 6
>> Z=PTGaussian(zeros(2,2,2)) % Multi-dimensional, too.
Z =
PTGaussian (2,2,2) (exact)
mean =
0* 0*
0* 0*
... more layers ...
```

Linear indexing

- Covariance matrix indexing uses *linear indices*, corresponding to column-major element storage.

```matlab
>> M
M =
1 2 3
4 5 6
>> M(:)
an =
1
4
2
5
3
6
```

- Thus, \( C(3,5) \) (and \( C(5,3) \)) correspond to the covariance between elements \( M(1,2) \) and \( M(1,3) \).

Reorganizing arrays

- Transpose `\(' and `.\)’. Does change the linear indices.
- `reshape` command. Does *not* change the linear indices.

```matlab
>> p'
PTGaussian (1,2)
mean = 3 4
>> reshape(A,3,2)
PTGaussian (3,2)
mean =
1 5
4 3
2 6
>> reshape(A,6,1) % Same as A(:)
PTGaussian (6,1)
mean =
std =
corr =
1 11
4 12
2 13
5 14
3 15
6 16
```
PTGaussian — A Matlab class for Multivariate Normally Distributed variables

PTGaussian arrays

Concatenation

Arrays may be stacked...

\[
\begin{align*}
\text{pq1} &= \left[ \begin{array}{c} p \\ q \end{array} \right] \quad \text{horizontally...} \\
\text{pq1} &= \text{PTGaussian}(2,2) \\
\text{mean} &= \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix} \\
\text{pq2} &= \left[ \begin{array}{c} p \\ q \end{array} \right] \quad \text{vertically...} \\
\text{pq2} &= \text{PTGaussian}(4,1) \\
\text{mean} &= \begin{bmatrix} 3 \\ 4 \\ 1.732 \\ 4 \\ 2.449 \\ 1.732 \\ 3 \\ 2.449 \end{bmatrix} \\
\text{pq3} &= \text{cat}(3,p,q) \quad \text{...and in any other direction...} \\
\text{pq3} &= \text{PTGaussian}(2,1,2) \\
\text{mean} &= \begin{bmatrix} 3 \\ 4 \end{bmatrix} \\

\end{align*}
\]

Example (2D points)

Independent Euclidean points may be stacked columnwise.

\[
\begin{align*}
\text{z} &= \left[ \begin{array}{c} p \\ q \\ s \end{array} \right] \\
\text{z} &= \text{PTGaussian}(2,3) \\
\text{mean} &= \begin{bmatrix} 3 & 6 & 5 \\ 4 & 5 & 3 \end{bmatrix} \\
\text{z} &= \text{PTGaussian}(6,1) \\
\text{mean} &= \begin{bmatrix} 3 \\ 1.732 \\ 4 \\ 2.449 \\ 6 \\ 1.732 \\ 5 \\ 1.732 \\ 5+1 \\ 3+1 \\ 3+1 \\ 1 \end{bmatrix} \\
\text{plot}(\text{z},[\text{pic}'])
\end{align*}
\]

Example (2D points)

Concatenated arrays are assumed to be independent.

\[
\begin{align*}
\text{pp} &= [p,p] \quad \text{horizontally...} \\
\text{pp} &= \text{PTGaussian}(4,1) \\
\text{mean} &= \begin{bmatrix} 3 \\ 1.732 \\ 4 \\ 2.449 \\ 1.732 \end{bmatrix} \\
\text{pp2} &= \text{repmat}(p,1,2) \quad \text{Same as p(:,[1,1])} \\
\text{pp2} &= \text{PTGaussian}(4,1) \\
\text{mean} &= \begin{bmatrix} 3+3 \\ 1.732 \\ 4+4 \\ 1.732 \end{bmatrix} \\
\end{align*}
\]

Use repmat or repeated index to generate dependent, repeated copies.

\[
\begin{align*}
\text{plot}(\text{z},[\text{pic}'])
\end{align*}
\]
PTGaussian — A Matlab class for Multivariate Normally Distributed variables

Subscripting

PTGaussian arrays can be subscripted as any array.

```
>> p(1)
PTGaussian (1,1)
mean = std = corr =
 3 1.732
>> A(1,2:3)' % Linear index [3,5].
PTGaussian (2,1)
mean = std = corr =
 2 13
>> A(2,:) % Linear index [2,4,6].
PTGaussian (1,3)
mean =
 4 5 6
>> A(5) % Linear index 5=element (1,3).
PTGaussian (1,1)
mean = std = corr =
 3 15
```

Subscripting and data access may be combined.

```
>> A(1,2:3).mean % Same as A.mean(1,2:3)
an =
 2 3
>> A(1,2:3).std % Same as A.std(1,2:3)
an =
 13 15
>> A(1,2:3).cov % Covariance matrix for linear indices [3,5].
an =
 169
 0
 0 225
>> A.cov(3,:) % Covariance between element 3 and all other.
an =
 0 0 0 0 0
```

Subscripted assignments works, too.

```
Right hand side (rhs) is assumed to be independent of lhs.

>> B=PTGaussian(ones(2,3),ones(6));
>> B(:,2:3)=p;p
B = PTGaussian (2,3)
mean =
 1+ 3 3
 1+ 4 4
>> B(:)
PTGaussian (6,1)
mean = std = corr =
 3 1.732
 4 2.449
```

Concatenated arrays are also assumed to be independent.

```
>> B=PTGaussian(ones(2,3),ones(6));
>> B(:,2:3)=[p;p]
B = PTGaussian (2,3)
mean =
 1+ 3 3
 1+ 4 4
>> B(:)
PTGaussian (6,1)
mean = std = corr =
 1+ 1
 1+ 1
 3 1.732
 4 2.449
```

Subscripted assignments

```
Subscripted assignments works, too.
Right hand side (rhs) is assumed to be independent of lhs.

>> B=PTGaussian(ones(2,3),ones(6));
>> B(:,3)=p
B = PTGaussian (2,3)
mean =
 1+ 3
 1+ 4
>> B(:)
PTGaussian (6,1)
mean = std = corr =
 3 1.732
 4 2.449
```

```
Use repeated arrays to transfer dependency.

```matlab
>> B=PTGaussian(ones(2,3),ones(6));
>> B(:,2:3)=p(:,[1,1]) % or B(:,2:3)=repmat(p,1,2)
```

```
B = PTGaussian (2,3)
mean =    1+ 3+ 3+
        1+ 4+ 4+
>> B(:)
```

```
PTGaussian (6,1)
mean = std = corr =
1+ 1     +++     ++++
1+ 1     +++     ++++
3+ 1.732 ++     +++     +++
4+ 2.449 ++     +++     +++
3+ 1.732 ++     +++     +++
4+ 2.449 ++     +++     +++
```

Example (Euclidean to homogenous coordinates)

Homogenous coordinates are extensions to Euclidean coordinates, where e.g. a 2D point is represented by a 3-vector

\[
v = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.
\]

All homogenous vectors that are equal up to multiplication by a non-zero scalar correspond to the same Euclidean point

\[
v = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = k \begin{pmatrix} x \\ y \\ k \end{pmatrix} = kv.
\]

Example (Translation)

Using homogenous coordinates, a translation of 2D points by \((d_x, d_y)^T\) is performed by the matrix

\[
T = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix},
\]

since

\[
Tv = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + d_x \\ y + d_y \\ 1 \end{pmatrix}.
\]
Example (Translation)

```matlab
T = PTGaussian(eye(3));
T(1:2,3) = q
v = T * [p;1]
p + q
```

Example (Camera calibration matrix)

```matlab
A camera calibration matrix $K$ for a digital pinhole camera with square pixels can be written as

$$
K = \begin{pmatrix}
    f & 0 & p_x \\
    0 & f & p_y \\
    0 & 0 & 1
\end{pmatrix}.
$$

Assume the vector $v = (p_x, p_y, f)^T$ has been estimated as

```matlab
v = PTGaussian(3,1)
```

Then the elements may be inserted in a camera calibration matrix as follows:

```matlab
K = PTGaussian(eye(3));
K([1,5,7,8]) = v([3,3,1:2])
```

Low level manipulation

The mean, variance, and covariance of individual elements can be directly manipulated.

```matlab
t = PTGaussian([1:4]', diag(5:8).^2);
t.mean(3) = 99
```
Warning!

- Manipulation of the covariance elements maintains symmetry but not positive semidefiniteness!

```matlab
>> t.var(2)=-1
PTGaussian (4,1)
mean = cov =
1 25 0 0 0
2 0 -1 0 0
99 0 0 49 0
4 0 0 0 64
>> t.cov(2,3:4)=[-33,-44]
PTGaussian (4,1)
mean = cov =
1 25 0 0 0
2 0 -1 -33 -44
99 0 0 49 0
4 0 0 0 64
```

The PTCoGaussian class

- The off-diagonal block may be stored in the PTCoGaussian helper class.

```matlab
>> m=[17.8 11.02 7.31]'; C=[100,2,4;2,4,3;4,3,4]/10000;
>> cam=PTGaussian(m,C)
cam = PTGaussian (3,1)
mean = std = corr =
17.8 0.1
11.02 0.02
7.31 0.02
>> [f,pp,fpp]=split(cam,1,2:3)
f = PTGaussian (1,1)
mean = std = corr =
17.8 0.1
pp = PTGaussian (2,1)
mean = std = corr =
11.02 0.02
7.31 0.02
fpp = PTGaussian (1-by-2)
```

Loss of covariance

- If a vector of values is split in two parts, the off-diagonal covariance blocks are lost.

\[
\begin{pmatrix}
  x \\
  y
\end{pmatrix}, \quad
C_{zz} = \begin{pmatrix}
  C_{xx} & C_{xy} \\
  C_{xy}^T & C_{yy}
\end{pmatrix}.
\]

```matlab
>> x=z(1:n);
>> y=z(n+1:end);
```

- The covariance and correlation values may still be accessed in the PTCoGaussian object.

```matlab
>> cam.cov(1,3)
an =
0.0004
>> fpp.cov(1,2)
an =
0.0004
>> max(abs(fpp.corr))
an =
0.2000
```

- The compound vector may be un-split by joining the components.

```matlab
>> cam2=join(f,pp,fpp)
cam2 = PTGaussian (3,1)
mean = std = corr =
17.8 0.1
11.02 0.02
7.31 0.02
```
Identity testing

- The standard == (equality) and ~= (inequality) tests are binary operators that test for identity.
- Two objects are identical if they have the same size, mean, and covariance.

```matlab
>> p==q
ans = 0
>> p==p
ans = 1
>> p~=p
ans = 0
```

The probability that the hypothesis is wrong is given by
\[ \Phi_n^{-1}(z^T C_{zz}^{-1} z). \]

```matlab
>> probdifferent(p,q)
ans = 0.5999
>> probdifferent(p-q,0)
ans = 0.5999
```

The corresponding test is given by

```matlab
>> isdifferent(p,q,0.95)
ans = 0
>> isdifferent(p,q,0.5)
ans = 1
```

Statistical testing

- To test if the n-vectors p and q are different, we set up the hypothesis
  \[ H_0 : p = q, \]  or equivalently
  \[ H_0 : z = 0, \]
  where \( z = p - q \).

```matlab
>> probdifferent(p,q)
ans = 0.5999
```

Basic error (variance) propagation

- Basic operations:
  \[ z = x + y, \quad C_{zz} = C_{xx} + C_{yy}, \]
  \[ z = x - y, \quad C_{zz} = C_{xx} + C_{yy}, \]
  \[ z = a x, \quad C_{zz} = a^2 C_{xx}. \]

```matlab
>> x=PTGaussian([1;1],diag([0.01,1]));
>> y=PTGaussian([1;0],diag([3,0.05]));
>> plot([x,y,x+y,x-y,1.5*x],'b-','r-','g--','m-.','k-.')
```

```matlab
>> x=PTGaussian([1;1],diag([0.01,1]));
>> y=PTGaussian([1;0],diag([3,0.05]));
>> plot([x,y,x+y,x-y,1.5*x],'b-','r-','g--','m-.','k-.')
```
PTGaussian — A Matlab class for Multivariate Normally Distributed variables

Error propagation

Basic error propagation

Warning!

All binary operations assume the operands are independent!

```matlab
>> pp=p+p % Dependency between terms is lost!
pp = PTGaussian (2,1)
mean = 6
std = 2.449
corr = +++
8
3.464
+++%
>>
```

```matlab
>> p2=2*p % Should be the same as pp
p2 = PTGaussian (2,1)
mean = 6
std = 3.464
corr = ++++
8
4.899
+++%
>>
```

```matlab
>> plot([pp,p2],sqrt(5.99),{'r','g'})
>> pi=sample(p,100); ppi=pi+pi;
>> hold on, plot(ppi(1,:),ppi(2,:),'o'); hold off
```

PTGaussian — A Matlab class for Multivariate Normally Distributed variables

Error propagation

Linear error propagation

Given a random variable

\[ x \sim N(\mu_x, C_{xx}) \]

and a linear transformation

\[ y = Ax + b, \]

the transformed variable is

\[ y \sim N(A\mu_x + b, AC_{xx}A^T), \]

i.e.

\[ \mu_y = A\mu_x + b, \]

\[ C_{yy} = AC_{xx}A^T. \]
Non-linear error propagation

- Given a non-linear function \( y = f(x) \) and its Taylor expansion

\[
y = \mu_y + d_y = f(\mu_x) + Jd_x + O(\|d_x\|^2),
\]

where the Jacobian is

\[
J = \begin{bmatrix} \frac{\partial f_i(x)}{\partial x_j} \end{bmatrix} \bigg|_{x = \mu_x},
\]

we get a first order approximation of the distribution of \( y \) as

\[
\mu_y = f(\mu_x),
\]

\[
\sigma_y = \sqrt{J\sigma^2_xJ^T} 
\approx 0.08.
\]

- Otherwise, the approximation is poor:

For

\[
y = \sin x, \\
\mu_x = \pi/6 \ (30^\circ), \\
\sigma_x = 5\pi/36 \ (25^\circ),
\]

we get

\[
\mu_y = \sin \pi/6 = 0.5, \\
J = \cos \pi/6, \\
\sigma_y = \sqrt{J\sigma^2_xJ^T} \approx 0.38.
\]
If the first order approximation is poor, a numerical re-sampling may be a better choice.

\[
\begin{align*}
&\text{if } \theta = \pi/6, \text{ then } y = \sin(x), \quad \text{for } x \in [0, \pi/3] \\
&\text{for } x \in [0, \pi/3], \quad \text{then } y = \sin(x), \quad \text{for } x \in [0, \pi/3]
\end{align*}
\]

If we have the implicit relationship

\[
f(x, y) = 0,
\]

we obtain the first order approximation

\[
df = J_x dx + J_y dy + J_z dz = 0.
\]

If \( J_x \) is invertible,

\[
dx = -J_x^{-1} J_y dy - J_x^{-1} J_z dz
\]

and

\[
C_{xx} = J_x^{-1} J_y C_{yy} J_y^T J_x^{-T} + J_x^{-1} J_z C_{zz} J_z^T J_x^{-T}.
\]
Again, the linearity of \( f(x, y) \) near \( \mu_x \pm k\sigma_x \), determines the quality of the approximation.

\[
\begin{align*}
\sigma_x &= 0.05 \\
\sigma_y &\approx 0.03
\end{align*}
\]

\[
\begin{align*}
\sigma_x &= 0.13 \\
\sigma_y &\approx 0.08
\end{align*}
\]

Given the expression

\[
f(A, x, b) = Ax - b = 0,
\]

with \( A \ m \times m \) and \( b \ m \times 1 \), we have

\[
J_x = A, \quad J_A = x^T \otimes I_m, \quad J_b = -I_m,
\]

where \( \otimes \) is the *Kronecker product*, and thus

\[
C_{xx} = A^{-1} \left( x^T \otimes I_m \right) C_{AA} \left( x \otimes I_m \right) A^{-T} + A^{-1} C_{bb} A^{-T}.
\]

With an exact \( A \), i.e. \( x = A^{-1}b \), we recover the linear transformation and the expected

\[
C_{xx} = A^{-1} C_{bb} A^{-T}.
\]

---

**Example (Homogenous to Euclidean coordinates)**

- Given a homogenous 3-vector, the Euclidean coordinates are recovered by division by the last element

\[
v = \begin{pmatrix} t \\ u \\ w \end{pmatrix} = \begin{pmatrix} t/w \\ u/w \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.
\]

- The function

\[
z = \begin{pmatrix} x \\ y \end{pmatrix} = f(v) = \begin{pmatrix} t/w \\ u/w \end{pmatrix}
\]

  has Jacobian

\[
J = \begin{pmatrix} 1/w & 0 & -t/w^2 \\ 0 & 1/w & -u/w^2 \end{pmatrix}.
\]

- Thus,

\[
C_{zz} = JC_{vv}J^T.
\]

---

**Example (Polar-to-cartesian conversion)**

- A vector with polar coordinates

\[
z = \begin{pmatrix} \theta \\ r \end{pmatrix}
\]

  has corresponding cartesian coordinates

\[
v = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \cos \theta \\ r \sin \theta \end{pmatrix}.
\]

- The transformation has Jacobian

\[
J = \begin{pmatrix} -r \sin \theta & \cos \theta \\ r \cos \theta & \sin \theta \end{pmatrix}
\]

  and

\[
C_{vv} = JC_{zz}J^T.
\]
Example (Polar-to-cartesian conversion)

```matlab
>> t=PTGaussian([0,30,60]*pi/180,1e-3);
>> r=PTGaussian([1,1,1],1e-2);
>> tr=[t;r], plot(tr),axis equal
tr = PTGaussian (2,3)
mean =
0 0.5236 1.047
1 1 1
>> xy=pol2cart(tr), plot(xy),axis equal
xy = PTGaussian (2,3)
mean =
1 0.866 0.5
0 0.5 0.866
```

Example (2D triangulation)

- Assume we have two cameras stations placed at positions \( p_1 \) and \( p_2 \) with measured bearings \( \theta_1 \) and \( \theta_2 \) to an unknown point \( x \).

This may be formulated as

\[
x = p_1 + \alpha_1 t_1, \quad \text{and} \quad x = p_2 + \alpha_2 t_2, \quad \text{where}
\]

\[
t_i = \begin{pmatrix}
\cos \theta_i \\
\sin \theta_i
\end{pmatrix}
\]

Reformulating,

\[
\begin{pmatrix}
-t_1 & 0 & l_1 \\
0 & -t_2 & l_2
\end{pmatrix}
\begin{pmatrix}
\alpha_1 \\
\alpha_2
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
= \begin{pmatrix}
p_1 \\
p_2
\end{pmatrix},
\]

the solution may be found from

\[
z = M^{-1} p.
\]
Example (2D triangulation)

For small errors ($\sigma_\theta=2^\circ$), the $z_b$ (green), $z$ (red), and numeric (black, dashed) approximations are almost equal. The $z_p$ (blue) estimation underestimates the propagated error.

For intermediate errors ($\sigma_\theta=5^\circ$), the $z_b$ (green) and $z_p$ (blue) estimations both underestimate the propagated error.

For large errors ($\sigma_\theta=11^\circ$), all first order estimates underestimate the propagated error, especially $z_b$ (green).
Linear models

- Assume we have a vector \( \mathbf{b} \) that is assumed to be an observation of a stochastic vector
  \[
  \mathbf{b} \sim \mathcal{N}(\tilde{\mathbf{b}}, \mathbf{C}_{bb}).
  \]
- Furthermore, we assume that the “exact” observation vector \( \tilde{\mathbf{b}} \) is explained by a linear model
  \[
  \mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}.
  \]
for some unknown value of the parameter vector \( \tilde{\mathbf{x}} \).
- The \( m \times n \)-matrix design matrix \( \mathbf{A} \), \( m \geq n \), is assumed to be of full rank.

The residual difference between the observations and what can be explained by the model is thus given by
\[
\mathbf{v} = \mathbf{b} - \tilde{\mathbf{b}} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \sim \mathcal{N}(0, \mathbf{C}_{bb}).
\]
If we choose to minimize the normalized residuals
\[
\Omega^2 = \mathbf{v}^T \mathbf{C}_{bb}^{-1} \mathbf{v} = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_{\mathbf{C}_{bb}^{-1}} = (\mathbf{b} - \mathbf{A}\mathbf{x})^T \mathbf{C}_{bb}^{-1} (\mathbf{b} - \mathbf{A}\mathbf{x}),
\]
we end up with the weighted normal equations
\[
\mathbf{A}^T \mathbf{W}\mathbf{A} \tilde{\mathbf{x}} = \mathbf{A}^T \mathbf{W}\mathbf{b},
\]
with
\[
\mathbf{W} = \mathbf{C}_{bb}^{-1}
\]
as the weight matrix.

The estimate \( \tilde{\mathbf{x}} \) of \( \mathbf{x} \) from the normal equations
\[
\tilde{\mathbf{x}} = \left(\mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{b},
\]
has variance
\[
\mathbf{C}_{\tilde{\mathbf{x}}} = \mathbf{M}\mathbf{C}_{bb}\mathbf{M}^T = \left(\mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{C}_{bb} \mathbf{C}_{bb}^{-1} \mathbf{A} \left(\mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{A}\right)^{-1}
= \left(\mathbf{A}^T \mathbf{C}_{bb}^{-1} \mathbf{A}\right)^{-1}.
\]

If the covariance of the observations is
\[
\mathbf{C}_{bb} = \sigma^2 \mathbf{S}_{bb},
\]
where \( \mathbf{S}_{bb} \) shows the covariance structure and \( \sigma^2 \) is unknown, we may still estimate \( \tilde{\mathbf{x}} \) with \( \mathbf{W} = \mathbf{S}_{bb}^{-1} \) since
\[
\tilde{\mathbf{x}} = \left(\mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{b} = \left(\frac{\sigma_0^2}{\sigma^2}\right) \left(\mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{b}
= \left(\mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{A}\right)^{-1} \mathbf{A}^T \mathbf{S}_{bb}^{-1} \mathbf{b}.
\]
Given $\hat{x}$ and $\hat{v} = b - A\hat{x}$, $\sigma_0^2$ may be estimated as

$$\hat{\sigma}_0^2 = \frac{\hat{v}^T S_{bb}^{-1} \hat{v}}{r},$$

where the redundancy $r$ is

$$r = m - n.$$

This enables us to estimate $C_{xx}$ as

$$\widehat{C}_{xx} = \hat{\sigma}_0^2 (A^T S_{bb}^{-1} A)^{-1}.$$

If $C_{bb} = S_{bb}$ is correctly assumed, $\sigma_0 = 1$ and $\sigma_0$ is known as the standard deviation of unit weight.

Otherwise, if e.g. $S_{bb} = I$, i.e. the observations are assumed to be independent, $\hat{\sigma}_0$ will be an estimate of the measurement error.

Example (2D triangulation, 3 stations)

- Adding a third station adds redundancy and should increase our knowledge.

- If the stations do not agree, the first order estimation (red) is still good.

Again, for intermediate angle errors ($\sigma_\theta=5^\circ$), the first order estimation (red) is good.
Summary

The PTGaussian class can be used to
- Simulate experiments.
- Do linear and non-linear transformations.
- Tests.
- More work is needed in estimation.

Future work

- Extend the PTGaussian class to handle non-linear estimation
  and Bundle adjustment.
- Use it in the The Photogrammetric Toolbox.

Research topics

- Estimation of rotation matrices.
  - In 2D, 4 elements, 3 constraints, 1 degree of freedom,
    \[ R = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \]
  - In 3D, 9 elements, 6 constraints, 3 degrees of freedom.
- Estimation of stereo rigs.
  - External uncertainty (position and orientation of rig in the world).
  - Internal uncertainty (relative orientation between the cameras).
- Selective (sparse) covariance matrices.