Real-Time Convex Optimization in Signal Processing

Evgeny Kovalev and Andrei Znobishchev
Group 5

Problem statement

Input:

- There's signal **x(t)** at time steps from t = 0 to t = num_of_time_steps;
- The linear dynamical system is defined as: **x(t+1) = Ax(t) + w(t)**;
- **w(t)** is independent identically distributed (IID) Gaussian noise N(0,W);
- **y(t) = Cx(t) + v(t)** - observation available to us at time step t;
- **v(t)** is IID N(0, V);

Goal:

- Estimate **x(t)** based on observations **y(t)**

Standard Kalman Filter - Optimization Formulation

- Current problem can be formulated as the following optimization problem (see References [1]):

$$\begin{array}{ll} \text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \text{subject to} & y_t = Cx + v_t, \end{array}$$

- With the measurement update:

$$\Sigma_{t|t} \quad = \quad \Sigma_{t|t-1} - \Sigma_{t|t-1} C^T \left(C \Sigma_{t|t-1} C^T + V \right)^{-1} C \Sigma_{t|t-1}$$

- And time update:

$$\hat{x}_{t+1|t} = A \hat{x}_{t|t}, \qquad \Sigma_{t+1|t} = A \Sigma_{t|t} A^T + W$$

Standard Kalman Filter - Optimization Formulation

- Current problem can be formulated as the following optimization problem (see References [1]):

available
at each
time step

$$\begin{array}{ll} \text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \text{subject to} & y_t = Cx + v_t, \end{array}$$

available
at each
time step

Standard Kalman Filter - Optimization Formulation

- Current problem can be formulated as the following optimization problem (see References [1]):

variables

$$
\begin{array}{ll}
\text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\
\text{subject to} & y_t = Cx + v_t,
\end{array}
$$

variables

1. Initialization of matrices and vectors

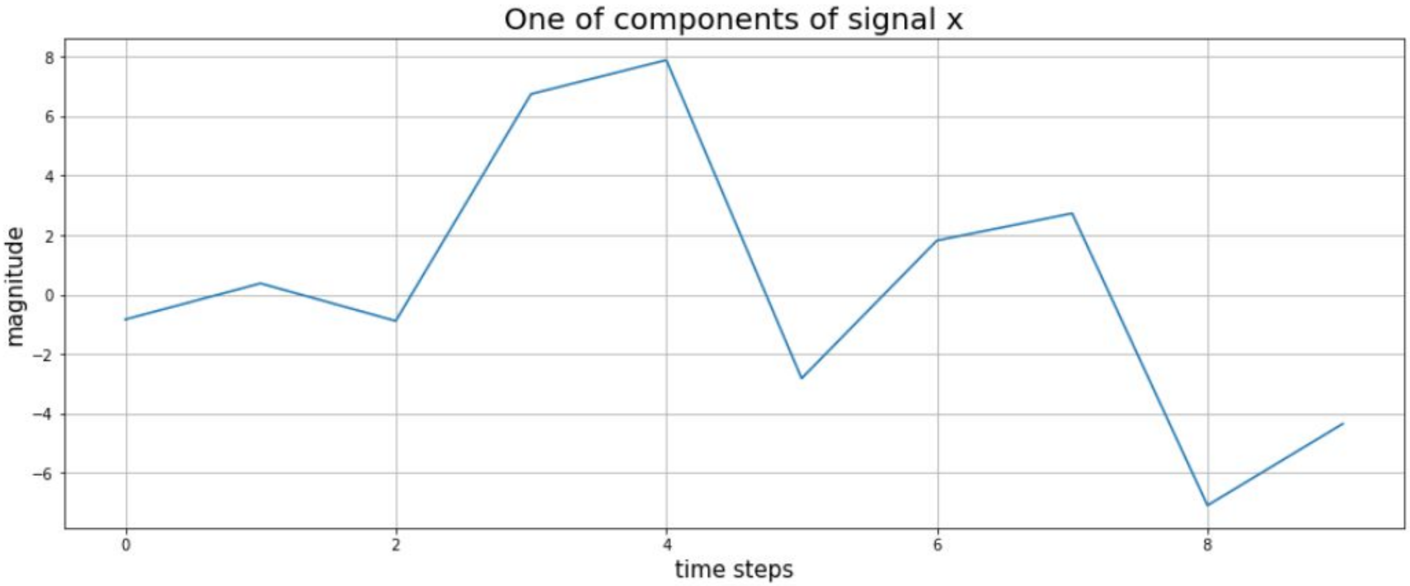
- There's signal **x(t)** at time steps from t = 0 to t = num_of_time_steps;
- The linear dynamical system is defined as: **x(t+1) = Ax(t) + w(t)**;
- **w(t)** is independent identically distributed (IID) Gaussian noise N(0,W);
- **y(t) = Cx(t) + v(t)** - observation available to us at time step t;
- **v(t)** is IID N(0, V);

In [2]:
```
N_s = 50
N_obs = 15

np.random.seed(10)
A = np.random.randn(N_s, N_s)
C = np.random.randn(N_obs, N_s)
max_mod = abs(max(np.linalg.eig(A)[0]))
A = A * 0.98 / max_mod

B = np.random.randn(N_s, 5)
W = B @ B.T
V = np.identity(N_obs)
v0 = np.random.multivariate_normal(np.zeros(N_obs), V)
w0 = np.random.multivariate_normal(np.zeros(N_s), W)
prob_mask = np.random.choice([0, 1], p=[0.95, 0.05], size=N_obs)

time_steps = 10
```

In [3]:
```
x_array = []
y_array = []
v_array = []
w_array = []
x0 = np.random.randn(N_s)
x_array.append(x0)
y0 = v0 + (1 - prob_mask) * (C @ x0)
y_array.append(y0)
v_array.append(v0)
w_array.append(w0)
for _ in range(time_steps - 1):
    prob_mask = np.random.choice([0, 1], p=[0.95, 0.05], size=N_obs)
    v_curr = np.random.multivariate_normal(np.zeros(N_obs), V)
    w_curr = np.random.multivariate_normal(np.zeros(N_s), W)
    x_prev = x_array[-1]
    x_curr = A @ x_prev + w_curr
    y_curr = v_curr + (1 - prob_mask) * (C @ x_curr)

    x_array.append(x_curr)
```

In [4]:
```
plt.figure(figsize=(16,6))
plt.title('One of components of signal x', fontsize=20)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(x_array[:, 0])
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7efcff77beb8>]
```



2.1 Standard Kalman filter (cvxpy solution)

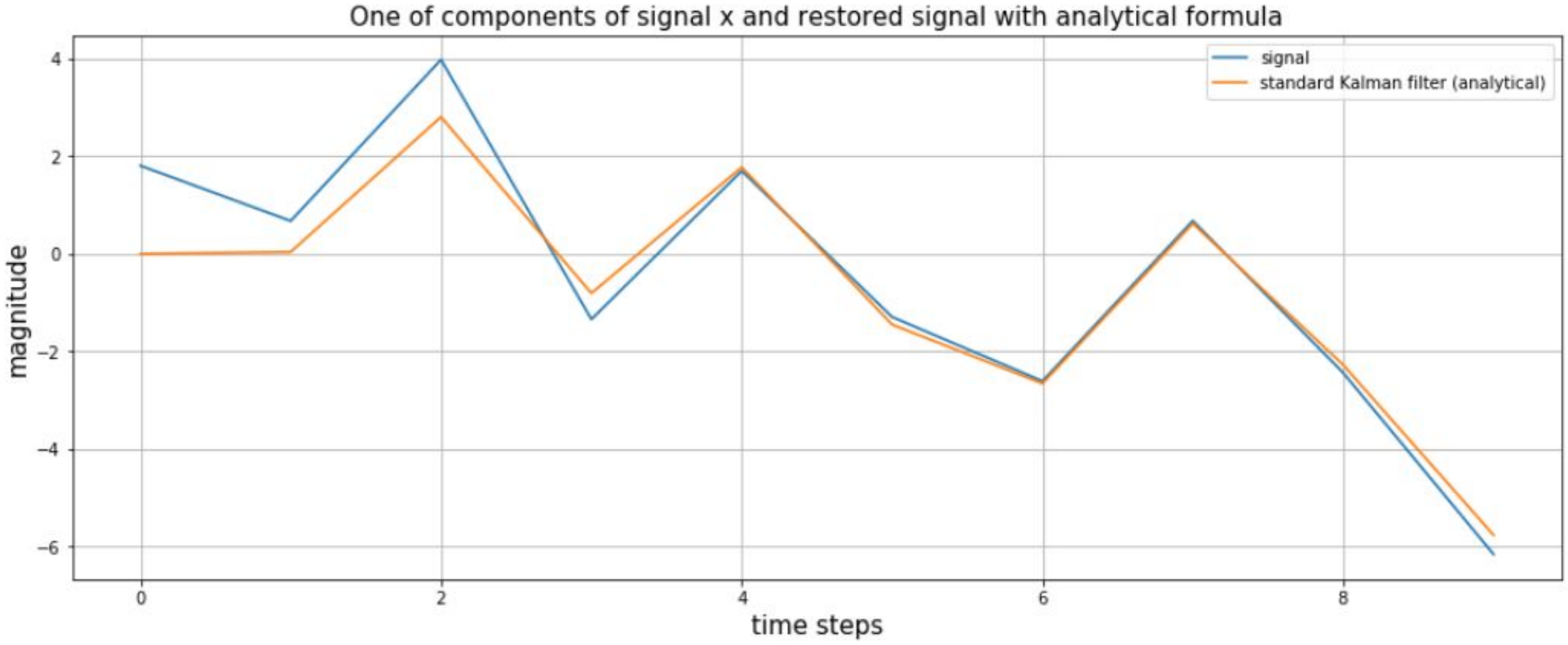
In [5]:
```
x_hat0 = np.zeros(N_s)
x_hat_array = np.array(x_hat0)
x_hat_array = x_hat_array.reshape((N_s,1))

sigma0 = np.identity(N_s)
sigma_array = np.array(sigma0)
sigma_array = sigma_array.reshape((N_s,N_s,1))

for i in tqdm(range(time_steps - 1)):
    # update steps
    x_hat_curr = x_hat_array[:,-1]
    sigma_curr = sigma_array[:,:,-1]
    x_hat_update = A @ x_hat_curr
    sigma_update = A @ sigma_curr @ A.T + W

    # formulate optimization problem
    x = cvx.Variable(N_s)
    v = cvx.Variable(N_obs)
    constraints = [y_array[i + 1] == C @ x + v]

    #P = np.linalg.inv(sigma_curr)
    P = np.linalg.inv(sigma_update)
    L = np.linalg.cholesky(P)
    objective = cvx.Minimize(cvx.norm(v)**2 + cvx.norm(L.T@(x - x_hat_update))**2)
    problem = cvx.Problem(objective, constraints)
    problem.solve(solver='SCS')

    x_hat_new = np.array(x.value)
    v_new = v.value

    x_hat_new = x_hat_new.reshape((N_s,1))
    x_hat_array = np.append(x_hat_array, x_hat_new, axis = 1)

    # for sigma the same
    #sigma_update = A @ sigma_curr @ A.T + W
    sigma_new = sigma_update  - sigma_update @ C.T @ np.linalg.inv(C @ sigma_update @ C.T + V) @ C @ sigma_update
    sigma_new = sigma_new.reshape((N_s,N_s,1))
    sigma_array = np.append(sigma_array, sigma_new, axis=2)
```

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t}, \qquad \Sigma_{t+1|t} = A\Sigma_{t|t}A^T + W$$

$$\begin{array}{ll} \text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \text{subject to} & y_t = Cx + v_t, \end{array}$$

$$\Sigma_{t|t} \;=\; \Sigma_{t|t-1} - \Sigma_{t|t-1}C^T \left(C\Sigma_{t|t-1}C^T + V\right)^{-1} C\Sigma_{t|t-1}$$

In [58]:
```
plt.figure(figsize=(16, 6))
plt.title('One of components of signal x and restored signal with analytical formula', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(x_array[:, 1], label='signal')
plt.plot(x_hat_array_an[:, 1], label='standard Kalman filter (analytical)')
plt.legend(loc='best')
plt.show()
```



Standard Kalman Filter

- Kalman Filter formulation:

$$\begin{array}{ll} \text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \text{subject to} & y_t = Cx + v_t, \end{array}$$

- Analytical solution exists:

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + \Sigma C^T (C \Sigma C^T + V)^{-1} (y_t - C\hat{x}_{t|t-1})$$

2.2 Standard Kalman filter (analytical solution)

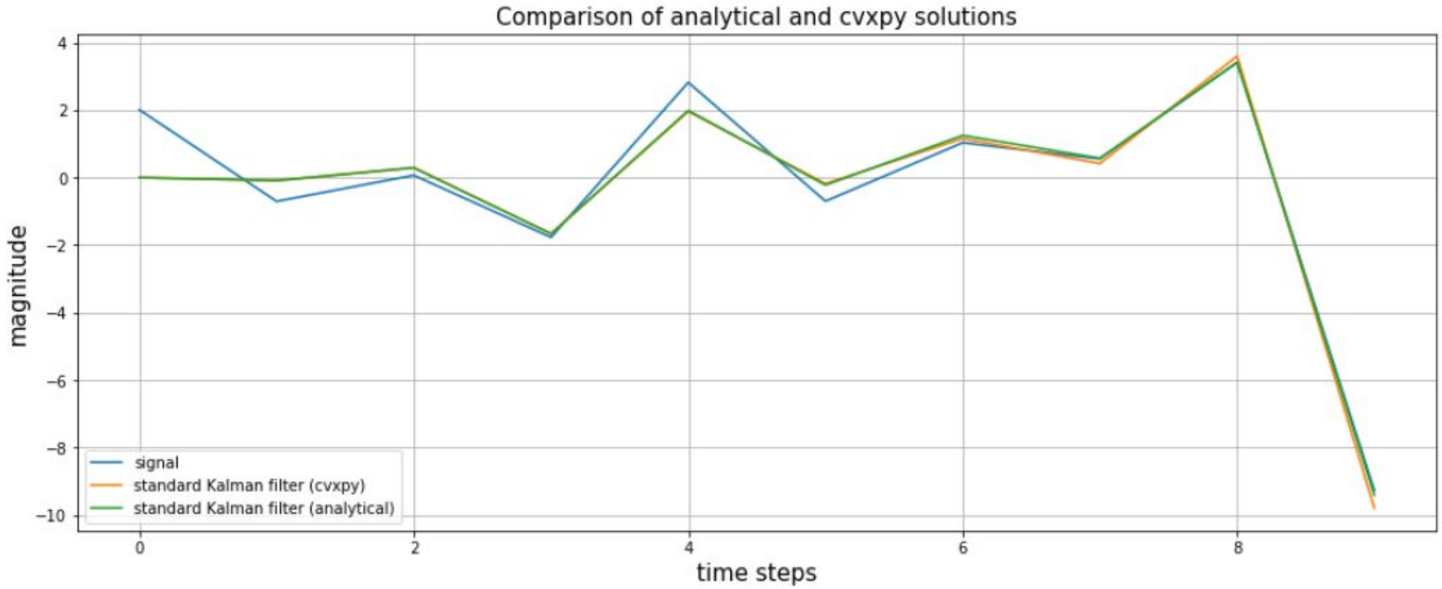
In [56]:
```
x_hat_array_an = []
x_hat0_an = np.zeros(N_s)
x_hat_array_an.append(x_hat0_an)

sigma_array_an = []
sigma0_an = np.identity(N_s)
sigma_array_an.append(sigma0_an)
```

In [57]:
```
for i in range(time_steps - 1):
    x_hat_curr_an = x_hat_array_an[-1]
    sigma_curr_an = sigma_array_an[-1]

    x_hat_update_an = A @ x_hat_curr_an
    sigma_update_an = A @ sigma_curr_an @ A.T + W
    #x_hat_new_an = x_hat_update_an + sigma_curr_an @ C.T @ np.linalg.inv(C @ sigma_curr_an @ C.T + V) @ (y_array[i +
    x_hat_new_an = x_hat_update_an + sigma_update_an @ C.T @ np.linalg.inv(C @ sigma_update_an @ C.T + V) @ (y_array[i
    x_hat_array_an.append(x_hat_new_an)

    #sigma_update_an = A @ sigma_curr_an @ A.T + W
    sigma_new_an = sigma_update_an  - sigma_update_an @ C.T @ np.linalg.inv(C @ sigma_update_an @ C.T + V) @ C @ sigma
    sigma_array_an.append(sigma_new_an)

x_hat_array_an = np.array(x_hat_array_an)
sigma_array_an = np.array(sigma_array_an)
```

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + \Sigma C^T (C \Sigma C^T + V)^{-1} (y_t - C\hat{x}_{t|t-1})$$

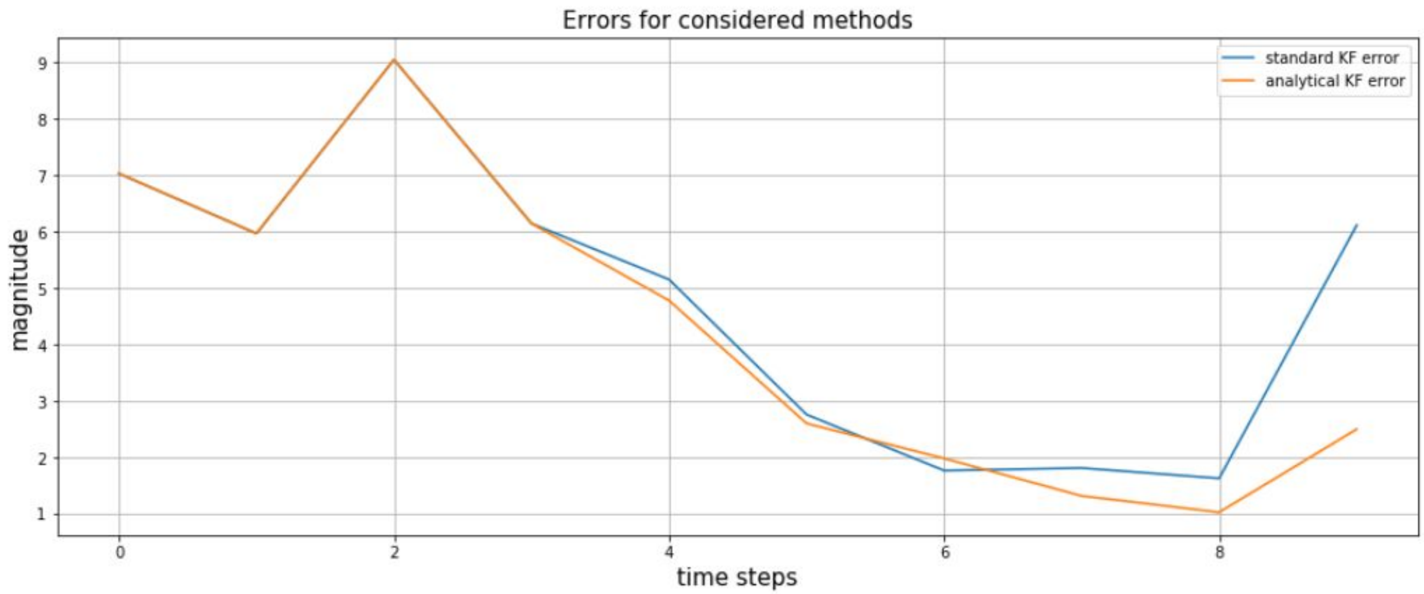
2.3. Comparison of analytical and cvxpy solutions

In [59]:
```
plt.figure(figsize=(16, 6))
plt.title('Comparison of analytical and cvxpy solutions', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(x_array[:, 10], label='signal')
plt.plot(x_hat_array[10,:], label='standard Kalman filter (cvxpy)')
plt.plot(x_hat_array_an[:, 10], label='standard Kalman filter (analytical)')
plt.legend(loc='best')
plt.show()
```



Comparison of CVXPY and analytical solutions

In [62]:
```
%matplotlib inline
fig = plt.figure(figsize=(16, 6))
plt.title('Errors for considered methods', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(np.linalg.norm(x_array.T-x_hat_array, axis=0), label='standard KF error')
plt.plot(np.linalg.norm(x_array.T-x_hat_array_an.T, axis=0), label='analytical KF error')
plt.legend(loc='best')
plt.show()
```



Robust Kalman Filter

Input:

- There's signal **$x(t)$** at time steps from $t = 0$ to t = num_of_time_steps;
- The linear dynamical system is defined as: **$x(t+1) = Ax(t) + w(t)$**;
- **$w(t)$** is independent identically distributed (IID) Gaussian noise $N(0,W)$;
- **$y(t) = Cx(t) + v(t) + z(t)$** - observation available to us at time step t;
- **$v(t)$** is IID $N(0, V)$;

Goal:

- Estimate **$x(t)$** based on observations **$y(t)$**

Robust Kalman Filter

Input:

- **$y(t) = Cx(t) + v(t) + z(t)$** - observation available to us at time step t;
- **$z(t)$** is an additional measurement noise term that is sparse. This term can be used to model unknown sensor failures, measurement outliers, or even intentional jamming;
- **$z(t)$** is defined as follows: it equals to 0 with probability 0.95, and it equals to $-Cx(t)$ with probability 0.05 (removes the signal)

Robust Kalman Filter - Optimization Formulation

$$\begin{array}{ll}\text{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ & \quad +\lambda \|z_t\|_1 \\ \text{subject to} & y_t = Cx + v_t + z_t,\end{array}$$

Robust Kalman Filter - Optimization Formulation

The term $\|x\|_1$ can written in element wise form:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

Then setting $|x_i| \leq t_i$ one could write:

$$\begin{aligned} \arg\min_{t} \quad & \mathbf{1}^T t \\ \text{subject to} \quad & Ax = b \\ & |x_i| \leq t_i \ \forall i \end{aligned}$$

Since $|x_i| \leq t_i \iff x_i \leq t_i,\ x_i \geq -t_i$ then:

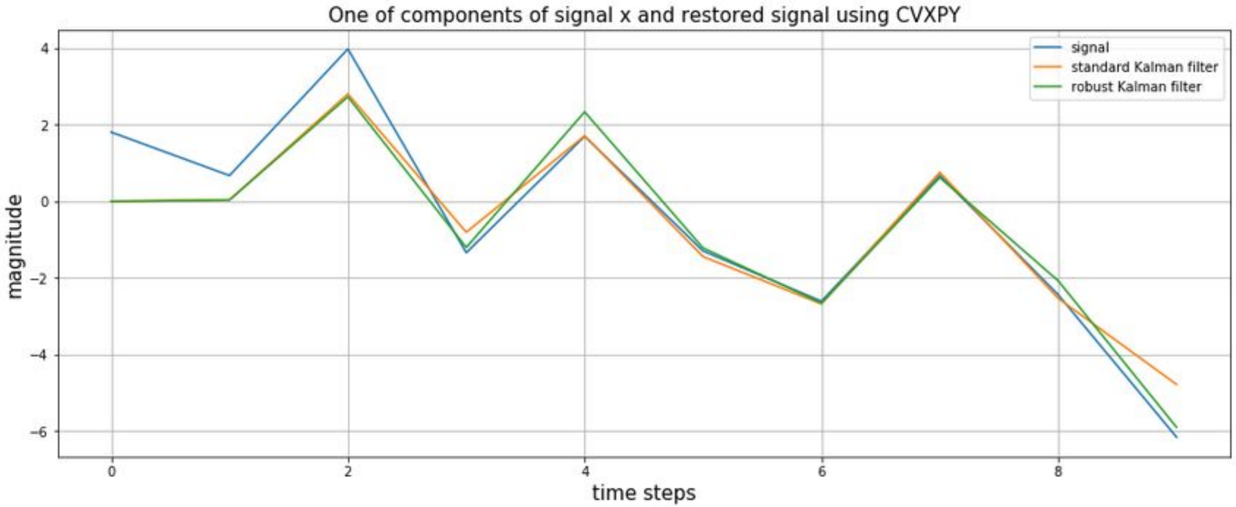
$$\begin{aligned} \arg\min_{t} \quad & \mathbf{1}^T t \\ \text{subject to} \quad & Ax = b \\ & x_i \leq t_i \ \forall i \\ & x_i \geq -t_i \ \forall i \end{aligned}$$

3. Robust Kalman filter

In [32]:
```
lam_array = np.array([2])#np.linspace(0.4, 0.43, num = 1)

for lam in lam_array:

    x_hat0_robust = np.zeros(N_s)
    x_hat_robust_array = np.array(x_hat0_robust)
    x_hat_robust_array = x_hat_robust_array.reshape((N_s,1))

    sigma0_robust = np.identity(N_s)
    sigma_robust_array = np.array(sigma0_robust)
    sigma_robust_array = sigma_robust_array.reshape((N_s,N_s,1))

    for i in tqdm(range(time_steps - 1)):
        # update steps
        x_hat_curr_robust = x_hat_robust_array[:,-1]
        sigma_curr_robust = sigma_robust_array[:,:,-1]

        x_hat_update_robust = A @ x_hat_curr_robust
        sigma_update_robust = A @ sigma_curr_robust @ A.T + W

        # formulate optimization problem
        x = cvx.Variable(N_s)
        v = cvx.Variable(N_obs)
        z = cvx.Variable(N_obs)
        t = cvx.Variable(N_obs)
        constraints = [y_array[i + 1] == C @ x + v + z,
                       lam*z <= t,
                       lam*z >=-t]

        P = np.linalg.inv(sigma_update_robust)
        L = np.linalg.cholesky(P)

        objective = cvx.Minimize(cvx.norm(v)**2 + cvx.norm(L.T @ (x - x_hat_update_robust))**2 + cvx.sum(t))
        problem = cvx.Problem(objective, constraints)
        problem.solve(solver='ECOS')
        x_hat_new_robust = x.value
        v_new = v.value
        z_new = z.value

        x_hat_new_robust = x_hat_new_robust.reshape((N_s,1))
        x_hat_robust_array = np.append(x_hat_robust_array, x_hat_new_robust, axis = 1)

        # for sigma the same
        #sigma_update_robust = A @ sigma_curr_robust @ A.T + W
        sigma_new_robust = sigma_update_robust  - sigma_update_robust @ C.T @ np.linalg.inv(C @ sigma_update_r
        sigma_new_robust = sigma_new_robust.reshape((N_s,N_s,1))
        sigma_robust_array = np.append(sigma_robust_array, sigma_new_robust, axis=2)
```

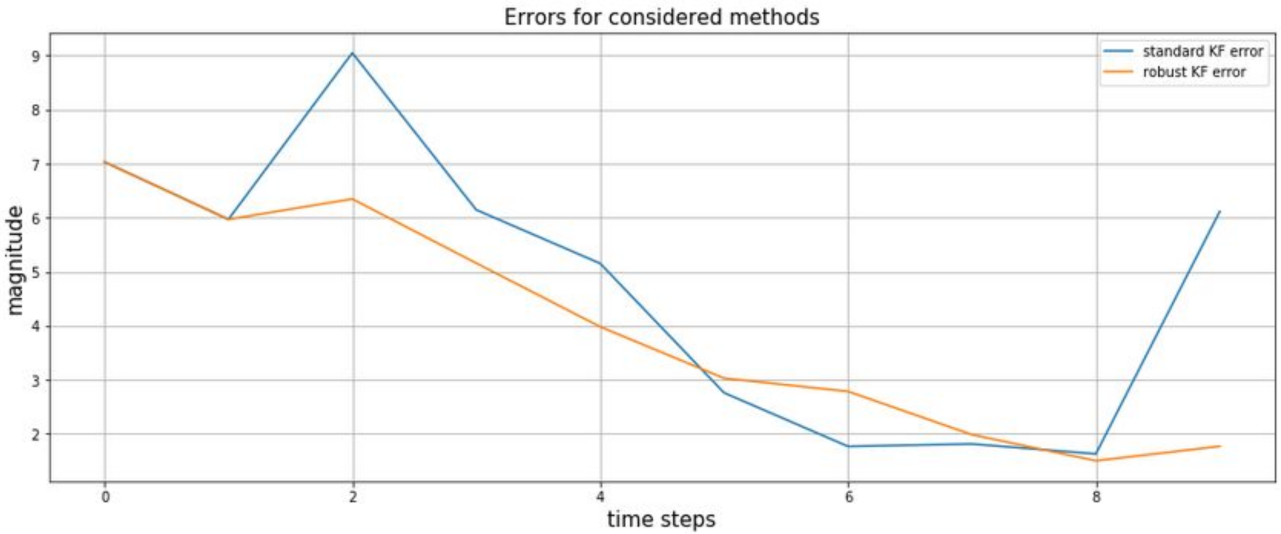
Robust Kalman Filter

In [33]:
```
%matplotlib inline
fig = plt.figure(figsize=(16, 6))
plt.title('One of components of signal x and restored signal using CVXPY', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(x_array[:, 1], label='signal')
plt.plot(x_hat_array[1, :], label='standard Kalman filter')
plt.plot(x_hat_robust_array[1, :], label='robust Kalman filter')
plt.legend(loc='best')
plt.show()
```



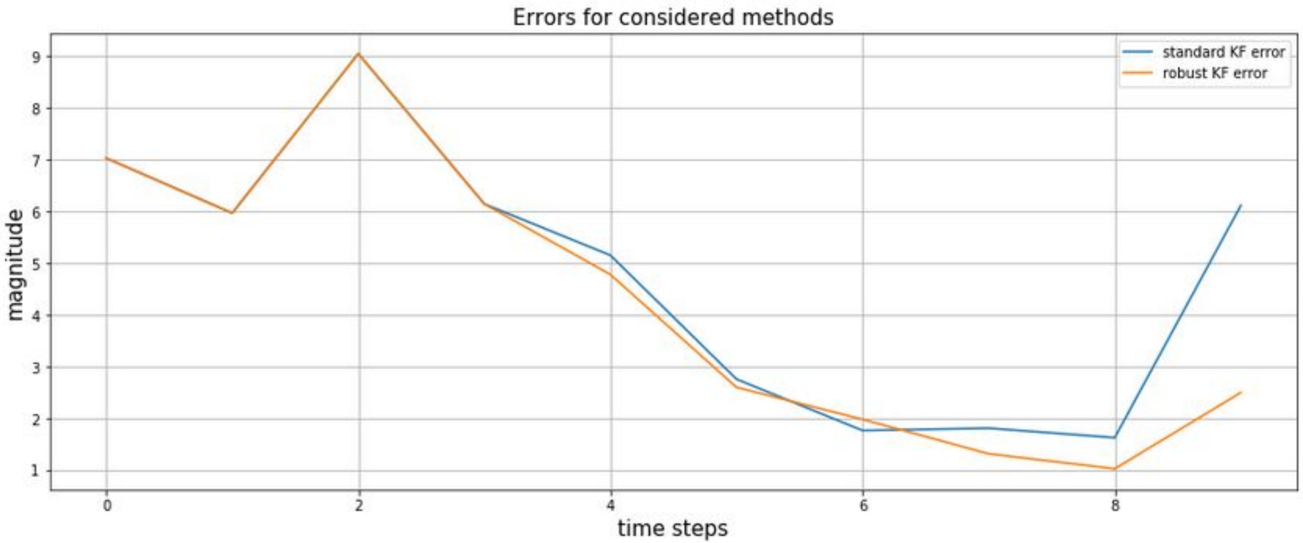
Robust Kalman Filter

In [34]:
```
%matplotlib inline
fig = plt.figure(figsize=(16, 6))
plt.title('Errors for considered methods', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(np.linalg.norm(x_array.T-x_hat_array, axis=0), label='standard KF error')
plt.plot(np.linalg.norm(x_array.T-x_hat_robust_array, axis=0), label='robust KF error')
plt.legend(loc='best')
plt.show()
```



Robust Kalman Filter (high lambda case)

In [71]:
```
%matplotlib inline
fig = plt.figure(figsize=(16, 6))
plt.title('Errors for considered methods', fontsize=15)
plt.xlabel('time steps', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.grid()
plt.plot(np.linalg.norm(x_array.T-x_hat_array, axis=0), label='standard KF error')
plt.plot(np.linalg.norm(x_array.T-x_hat_robust_array, axis=0), label='robust KF error')
plt.legend(loc='best')
plt.show()
```



Summary and Possible Applications

- Standard Kalman Filter problem was solved in CVXPY as a QP optimization problem;
- Kalman Filtering can be used for a linear dynamical system driven by Gaussian noise;
- Analytical solution supported the correctness of the CVXPY solution;
- Robust Kalman Filter problem was transformed to QP and solved using CVXPY;
- RKF can provide better results of signal restoration if part of noise is expected to model unknown sensor failures, measurement outliers, or intentional jamming

References

[1] J. Mattingley and S. Boyd, Real-Time Convex Optimization in Signal Processing, IEEE Signal Processing Magazine, 27(3):50-61, May 2010

[2] S.Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004

[3] http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf - Kalman Filter Derivation

Thank you for your attention!