DS-GA 3001.009 Modeling Time Series Data

Week 3 Kalman Filter

In [1]:
```
# Install PyKalman
# pip install pykalman
import numpy as np
import matplotlib.pyplot as plt
from pykalman import KalmanFilter
from scipy.stats import multivariate_normal

# Data Visualiztion
def plot_kalman(x,y,nx,ny,kx=None,ky=None, plot_type="r-", label=None):
    """
    Plot the trajectory
    """
    fig = plt.figure()
    if kx is not None and ky is not None:
        plt.plot(x,y,'g-',nx,ny,'b.',kx,ky, plot_type)
        plt.plot(kx[0], ky[0], 'or')
        plt.plot(kx[-1], ky[-1], 'xr')
    else:
        plt.plot(x,y,'g-',nx,ny,'b.')

    plt.xlabel('X position')
    plt.ylabel('Y position')
    plt.title('Parabola')

    if kx is not None and ky is not None and label is not None:
        plt.legend(('true','measured', label))
    else:
        plt.legend(('true','measured'))

    return fig

def visualize_line_plot(data, xlabel, ylabel, title):
    """
    Function that visualizes a line plot
    """
    plt.plot(data)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()

def print_parameters(kf_model, need_params=None):
    """
    Function that prints out the parameters for a Kalman Filter
    @param - kf_model : the model object
    @param - need_params : a list of string
    """
    if need_params is None:
        need_params = ['transition_matrices', 'observation_matrices', 't
ransition_offsets',
                      'observation_offsets', 'transition_covariance',
                      'observation_covariance', 'initial_state_mean', 'initi
al_state_covariance']
    for param in need_params:
        print("{0} = {1}, shape = {2}\n".format(param, getattr(kf_model,
param), getattr(kf_model, param).shape))
```

Data

We will use a common physics problem with a twist. This example will involve firing a ball from a cannon at a 45-degree angle at a velocity of 100 units/sec. We have a camera that will record the ball's position (pos_x, pos_y) from the side every second. The positions measured from the camera $(\hat{pos}_x, \hat{pos}_y)$ have significant measurement error.

Latent Variable $z = [pos_x, pos_y, V_x, V_y]$

Observed Variable $x = [\hat{pos}_x, \hat{pos}_y, \hat{V}_x, \hat{V}_y]$

Reference: [http://greg.czerniak.info/guides/kalman1/ (http://greg.czerniak.info/guides/kalman1/)](http://greg.czerniak.info/guides/kalman1/)

In [2]:

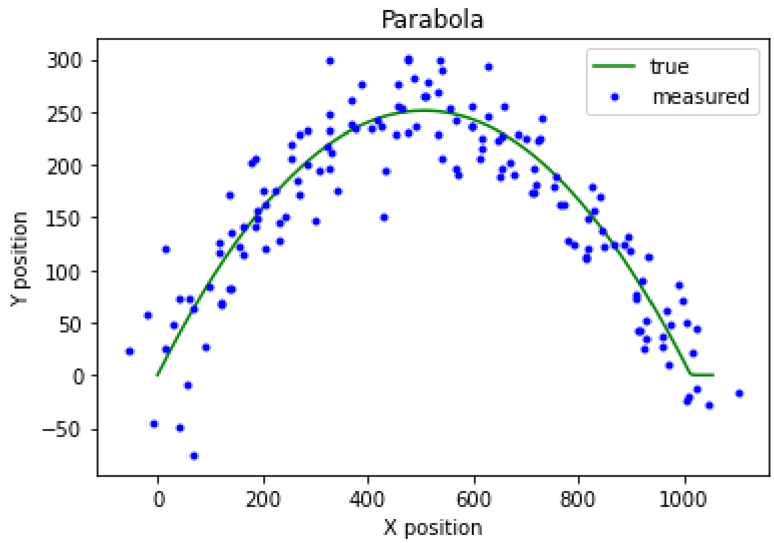
```
# true (latent) trajectory
x = [0, 7.0710678118654755, 14.142135623730951, 21.213203435596427, 28.2
84271247461902, 35.35533905932738, 42.42640687119285, 49.49747468305833,
56.568542494923804, 63.63961030678928, 70.71067811865476, 77.78174593052
023, 84.8528137423857, 91.92388155425118, 98.99494936611666, 106.0660171
7798213, 113.13708498984761, 120.20815280171308, 127.27922061357856, 13
4.35028842544403, 141.4213562373095, 148.49242404917499, 155.56349186104
046, 162.63455967290594, 169.7056274847714, 176.7766952966369, 183.84776
310850236, 190.91883092036784, 197.9898987322333, 205.0609665440988, 21
2.13203435596427, 219.20310216782974, 226.27416997969522, 233.3452377915
607, 240.41630560342617, 247.48737341529164, 254.55844122715712, 261.629
5090390226, 268.70057685088807, 275.77164466275354, 282.842712474619, 28
9.9137802864845, 296.98484809834997, 304.05591591021545, 311.12698372208
09, 318.1980515339464, 325.2691193458119, 332.34018715767735, 339.411254
9695428, 346.4823227814083, 353.5533905932738, 360.62445840513925, 367.6
955262170047, 374.7665940288702, 381.8376618407357, 388.90872965260115,
395.9797974644666, 403.0508652763321, 410.1219330881976, 417.19300090006
305, 424.26406871192853, 431.335136523794, 438.4062043356595, 445.477272
14752496, 452.54833995939043, 459.6194077712559, 466.6904755831214, 473.
7615433949868, 480.8326112068523, 487.9036790187178, 494.9747468305833
, 502.04581464244876, 509.11688245431424, 516.1879502661798, 523.2590180
780453, 530.3300858899108, 537.4011537017764, 544.4722215136419, 551.543
2893255074, 558.614357137373, 565.6854249492385, 572.756492761104, 579.8
275605729696, 586.8986283848351, 593.9696961967006, 601.0407640085662, 6
08.1118318204317, 615.1828996322972, 622.2539674441628, 629.325035256028
3, 636.3961030678938, 643.4671708797594, 650.5382386916249, 657.60930650
34904, 664.680374315356, 671.7514421272215, 678.822509939087, 685.893577
7509525, 692.9646455628181, 700.0357133746836, 707.1067811865491, 714.17
78489984147, 721.2489168102802, 728.3199846221457, 735.3910524340113, 74
2.4621202458768, 749.5331880577423, 756.6042558696079, 763.6753236814734
, 770.7463914933389, 777.8174593052045, 784.88852711707, 791.95959492893
55, 799.0306627408011, 806.1017305526666, 813.1727983645321, 820.2438661
763977, 827.3149339882632, 834.3860018001287, 841.4570696119943, 848.528
1374238598, 855.5992052357253, 862.6702730475909, 869.7413408594564, 87
6.8124086713219, 883.8834764831875, 890.954544295053, 898.0256121069185,
905.096679918784, 912.1677477306496, 919.2388155425151, 926.309883354380
6, 933.3809511662462, 940.4520189781117, 947.5230867899772, 954.59415460
18428, 961.6652224137083, 968.7362902255738, 975.8073580374394, 982.8784
258493049, 989.9494936611704, 997.020561473036, 1004.0916292849015, 101
1.162697096767, 1018.2337649086326, 1025.304832720498, 1032.375900532363
6, 1039.4469683442292, 1046.5180361560947, 1053.5891039679602]
y = [0, 6.972967811865475, 13.847835623730951, 20.624603435596427, 27.30
327124746l904, 33.88383905932738, 40.366306871192855, 46.750674683058335
, 53.036942494923814, 59.22511030678929, 65.31517811865477, 71.307145930
52025, 77.20101374238573, 82.9967815542512, 88.69444936611667, 94.294017
17798214, 99.79548498984762, 105.1988528017131, 110.50412061357858, 115.
71128842544405, 120.82035623730953, 125.831324049175, 130.74419186104046
, 135.55895967290593, 140.2756274847714, 144.89419529663687, 149.4146631
0850234, 153.83703092036782, 158.1612987322333, 162.38746654409877, 166.
51553435596423, 170.5455021678297, 174.4773699796952, 178.31113779156067
, 182.04680560342615, 185.68437341529162, 189.2238412271571, 192.6652090
3902256, 196.00847685088803, 199.25364466275352, 202.40071247461898, 20
5.44968028648447, 208.40054809834993, 211.2533159102154, 214.00798372208
087, 216.6645515339463 4, 219.22301934581182, 221.6833871576773, 224.0456
5496954277, 226.30982278140823, 228.4758905932737, 230.54385840513916, 2
32.51372621700463, 234.3854940288701, 236.1591618407355 8, 237.8347296526
```

```
0106, 239.41219746446652, 240.891565276332, 242.27283308819744, 243.5560
009000629, 244.7410687119284, 245.82803652379386, 246.81690433565933, 24
7.7076721475248, 248.50033995939026, 249.19490777125574, 249.79137558312
12, 250.2897433949867, 250.69001120685215, 250.99217901871762, 251.19624
683058308, 251.30221464244855, 251.31008245431403, 251.2198502661795, 25
1.03151807804497, 250.74508588991043, 250.3605537017759, 249.87792151364
138, 249.29718932550685, 248.61835713737233, 247.8414249492378, 246.9663
9276110326, 245.99326057296872, 244.9220283848342, 243.75269619669967, 2
42.48526400856514, 241.11973182043062, 239.65609963229608, 238.094367444
16155, 236.434535256027, 234.67660306789247, 232.82057087975795, 230.866
43869162342, 228.8142065034889, 226.66387431535435, 224.41544212721982,
222.0689099390853, 219.62427775095077, 217.08154556281625, 214.440713374
6817, 211.70178118654718, 208.86474899841264, 205.9296168102781, 202.896
3846221436, 199.76505243400905, 196.53562024587453, 193.20808805773999,
189.78245586960546, 186.2587236814709, 182.63689149333638, 178.916959305
20186, 175.09892711706732, 171.1827949289328, 167.16856274079825, 163.05
623055266372, 158.84579836452917, 154.53726617639464, 150.13063398826012
, 145.62590180012558, 141.02306961199105, 136.3221374238565, 131.5231052
3572197, 126.62597304758744, 121.6307408594529, 116.53740867131836, 111.
34597648318382, 106.05644429504929, 100.66881210691476, 95.1830799187802
2, 89.59924773064569, 83.91731554251115, 78.1372833543766, 72.2591511662
4206, 66.28291897810752, 60.208586789972976, 54.03615460183844, 47.76562
24137039, 41.39699022556936, 34.930258037434825, 28.365425849300287, 21.
70249366116575, 14.941461473031213, 8.082329284896677, 1.125097096762139
9, 0, 0, 0, 0, 0, 0]
# observed (noisy) trajectory
nx = [-55.891836789860065, -8.619869715037396, 42.294527931003934, -19.2
82331191905236, 15.680071645375804, 69.254448170858, 89.33867920263654,
28.666899505436437, 15.757974418210033, 56.95110872477952, 119.042464976
36771, 61.62441951678902, 39.29934181599181, 138.7343828583496, 96.51963
541398798, 117.86368373222598, 67.09014965331974, 135.85134345741767, 12
1.0077151824216, 162.7264612213607, 118.09485999718689, 161.385162314855
7, 153.61801317724127, 138.04480650773763, 136.57674149045124, 186.64610
285009547, 190.4353428154434, 187.08542653674456, 179.34131017012345, 20
3.60872317676595, 230.44206917637285, 200.5851735051386, 188.42133059532
134, 232.7400979104982, 225.30478119739485, 267.2756895668691, 203.85938
526332433, 283.3059183111086, 243.60174631799248, 255.04877235779932, 29
8.9707904316639, 269.8143043032952, 252.32059784738885, 270.558913570391
84, 326.59494694358125, 327.44121025500505, 323.8788885796068, 342.89989
39277302, 284.8080106831782, 308.30384319438355, 331.602751441996, 369.3
7448486134394, 326.7534721557288, 432.3093248477627, 406.1850938379638,
368.3192718477926, 417.1242042573391, 376.80066629284977, 328.0704042595
63, 453.310935909394, 428.1929004969645, 387.52306470654355, 455.6351576
860221, 455.4826886731417, 425.56091814116144, 464.50297459897234, 473.8
3062985528187, 508.0018220206407, 538.1473047813328, 474.1768111706383,
487.07340663957956, 513.8908741322764, 504.02223767836966, 533.566618799
3398, 475.84484473401085, 540.774394416208, 532.450138885039, 596.197329
3350309, 616.4286469911126, 571.2502050174375, 490.3795627036758, 595.11
92153241902, 565.0843289032891, 565.8861393244426, 535.3381668961649, 55
4.494293004607, 614.8000696382559, 610.6070255172268, 595.1137023505969,
655.8549550650116, 652.1364054115451, 627.0822281047308, 684.84591808309
78, 627.4287726057535, 646.4232572721058, 649.8096669791199, 713.6046684
169744, 698.0070989016643, 670.0057860355556, 652.4216425995312, 676.104
8509824369, 716.7538322065785, 755.4377489437768, 721.282075245216, 710.
0104845231749, 727.173927219191, 728.0409503769652, 765.4419988280144, 7
14.4351907609439, 780.131570018707, 788.9558247395082, 826.4452430146916
, 753.5313843631791, 772.7596204233874, 816.0768173822422, 827.278330906
```

```
8349, 813.59413380325, 868.0575409685421, 819.297427582733, 842.46574162
22495, 886.9452546786852, 848.0810151733698, 840.3562201041242, 814.5238
953430719, 909.815296516205, 932.007124918161, 920.1521325921302, 894.73
81254084694, 898.8234101267616, 916.4237616794475, 909.8509105807899, 95
9.775584291071, 913.833379291161, 928.6214973254758, 1023.4704479915798,
972.8771173177097, 929.6006265282693, 922.4096220769878, 965.64993444686
68, 1004.1043594740715, 998.0662730766439, 990.0011025721882, 1003.11927
78598809, 958.761271165956, 1014.5513285266792, 1102.4832378048654, 100
9.1078733361437, 1045.4004495002835, 969.8376930757283, 1023.92091084078
33]
ny = [23.580712916615695, -45.62854499965875, -48.454167220387774, 57.63
682593281395, 24.958737249448383, -75.44208650853136, 26.41512511354887,
49.05382536552557, 119.5753276811912, -8.883926248926471, 68.61653476275
158, 73.70930930561039, 73.33534997643633, 136.46569500850046, 84.651571
43902442, 116.75120295727146, 64.03491049317876, 83.17721636270467, 67.4
8086927886698, 114.0031414926982, 126.98883515540626, 141.94465033561954
, 121.4932906407089, 82.17340012376054, 171.20366262408706, 140.86199817
266035, 156.15800993843771, 205.18083992090502, 201.76590950501574, 120.
00689401909713, 127.22410828056267, 174.70956131509004, 148.236566537492
7, 145.51245102878624, 176.1302017047836, 185.20777488199408, 162.601891
6409635, 201.10110170021446, 151.66614689027648, 219.42574853381643, 14
6.53360660925895, 172.0394862006801, 205.89640027275198, 228.51657142871
176, 233.09734143057807, 196.9051845126125, 217.61161160052148, 176.2328
457551726, 232.91906774032418, 194.05143658995405, 212.54799704705485, 2
37.58982040493999, 248.06683550872324, 194.1313367493484, 234.1091445570
8777, 260.9183828690817, 241.38145360438665, 234.18839201094602, 298.715
6304059963, 229.37945966087958, 150.42222803196753, 276.519008189128, 25
4.65250291987843, 276.76968161072347, 236.66608705629918, 252.7664567709
944, 300.2402250565306, 265.79289722004, 206.71833433177704, 230.8006442
2085013, 282.21969317504124, 278.7505962073449, 264.65420420047275, 269.
1414531557679, 300.78849412505315, 290.40324997636577, 228.4884963762248
, 255.06136385539696, 224.79305441888243, 190.58455654967446, 236.931677
8500026, 237.1642802017231, 197.04061689021333, 243.08831859311766, 298.
69768812880136, 253.7179052734462, 214.9052914066147, 206.4070815355148,
236.37282969584498, 256.212452196932, 227.34374697096126, 245.9607452631
2518, 228.28624137197147, 293.80645840131524, 223.3350286792413, 189.114
90296131603, 173.18703394374444, 224.19402293984098, 201.5893028879243,
196.71922438575157, 191.29908495274748, 181.68302817527396, 189.52748698
83944, 223.8798616894293, 174.03835368022368, 225.8029667750153, 243.329
63834156004, 161.72407829159798, 195.87525096604634, 128.4947824255718,
125.03030538969553, 178.94162993285272, 178.69848164174897, 163.25298077
414635, 120.14847616048053, 156.19072198888165, 111.86292881870203, 123.
53293860987199, 149.80320169931565, 137.76193157426053, 123.562914124110
45, 122.88482052173731, 169.4024959581775, 113.64420024942245, 75.953589
36834089, 112.73548045966685, 89.77944981081296, 132.3832257220346, 118.
28500887622158, 41.80194386445528, 72.97395976865779, 27.564707182159275
, 41.944224628335874, 35.26542969310611, 43.441159853568024, 48.78418131
0168904, 52.156577913216324, 25.654795701109094, 61.6957571291634, 50.60
70208231052, 71.1942426060467, 85.42901900020982, -23.71494481685784, 3
6.536539251335405, 21.97545137095642, -16.352874522717297, -20.656788404
218812, -27.959598326260984, 10.981406961205966, -12.446671920156849]
data = np.array([nx,ny]).T
```

In [3]:
```
print(data.shape)
_ = plot_kalman(x,y,nx,ny);
```

```
(150, 2)
```



Review on Gaussian marginal and conditional distributions

Assume

$$z = [x^T y^T]^T$$
$$z = \begin{bmatrix} x \\ y \end{bmatrix} \sim N\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}\right)$$

then the marginal distributions are

$$x \sim N(a, A)$$
$$y \sim N(b, B)$$

and the conditional distributions are

$$x|y \sim N(a + CB^{-1}(y - b), A - CB^{-1}C^T)$$
$$y|x \sim N(b + C^T A^{-1}(x - a), B - C^T A^{-1}C)$$

important take away: given the joint Gaussian distribution we can derive the conditionals

Review on Linear Dynamical System

Latent variable:

$$z_n = Az_{n-1} + w$$

Observed variable:

$$x_n = Cz_n + v$$

Gaussian noise terms:

$$w \sim N(0, \Gamma)$$
$$v \sim N(0, \Sigma)$$
$$z_0 \sim N(\mu_0, \Gamma_0)$$

As a consequence, z_n, x_n and their joint distributions are Gaussian so we can easily compute the marginals and conditionals.

<img src='img/LDS.svg', width = 300, height=300>

right now n depends only on what was one time step back $n-1$ (Markov chain)

Given the graphical model of the LDS we can write out the joint probability for both temporal sequences:

$$P(\mathbf{z}, \mathbf{x}) = P(z_0) \prod_{n=1...N} P(z_n | z_{n-1}) \prod_{n=0...N} P(x_n | z_n)$$

all probabilities are implicitely conditioned on the parameters of the model

Kalman

We want to infer the latent variable z_n given the observed variable x_n.

$$P(z_n | x_1, \ldots, x_n, x_{n+1}, \ldots, x_N) \sim N(\hat{\mu}_n, \hat{V}_n)$$

Forward: Filtering

obtain estimates of latent by running the filtering from $n = 0, \ldots . N$

prediction given latent space parameters

<img src='img/LDS_latent.svg', width = 110, height=90>

$$z_n^{pred} \sim N(\mu_n^{pred}, V_n^{pred})$$
$$\mu_n^{pred} = A\mu_{n-1}$$

this is the prediction for z_n obtained simply by taking the expected value of z_{n-1} and projecting it forward one step using the transition probability matrix A

$$V_n^{pred} = AV_{n-1}A^T + \Gamma$$

same for the covariance taking into account the noise covariance Γ

correction (innovation) from observation

<img src='img/LDS_observed.svg', width = 40, height=80>

project to observational space:

$$x_n^{pred} \sim N(C\mu_n^{pred}, CV_n^{pred}C^T + \Sigma)$$

correct prediction by actual data:

$$z_n^{innov} \sim N(\mu_n^{innov}, V_n^{innov})$$
$$\mu_n^{innov} = \mu_n^{pred} + K_n(x_n - C\mu_n^{pred})$$
$$V_n^{innov} = (I - K_nC)V_n^{pred}$$

Kalman gain matrix:

$$K_n = V_n^{pred}C^T(CV_n^{pred}C^T + \Sigma)^{-1}$$

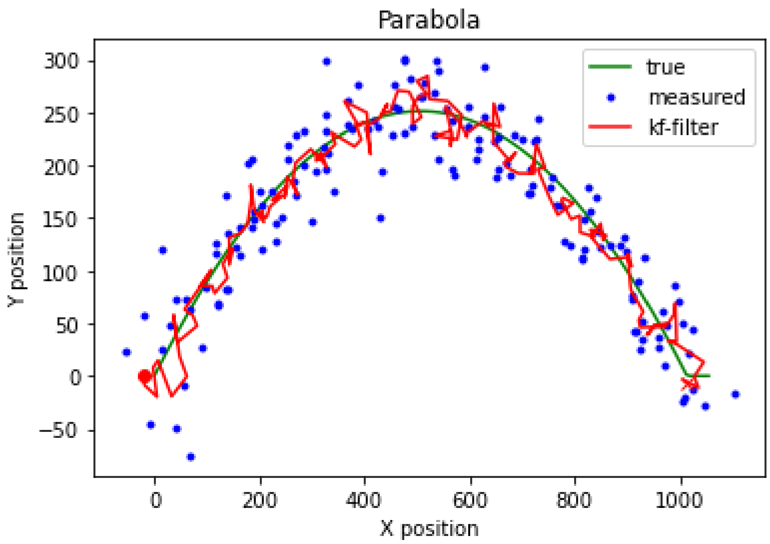
we use the latent-only prediction to project it to the observational space and compute a correction proportional to the error $x_n - CAz_{n-1}$ between prediction and data, coefficient of this correction is the Kalman gain matrix

<img src='img/Kfilter_Bishop.png', width = 600, height=600> from Bishop (2006), chapter 13.3

if measurement noise is small and dynamics are fast -> estimation will depend mostly on observed data

Kalman Filter to predict true (latent) trajectory from observed variable using Pykalman API

In [4]:
```
kf = KalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1])
# fit the model (use EM algorithm to estimate the parameters, we will no
t worry about this for now)
kf.em(data, n_iter=6)
# Kalman filtering
filtered_state_means, filtered_state_covariances = kf.filter(data)
fig = plot_kalman(x,y,nx,ny, filtered_state_means[:,0], filtered_state_m
eans[:,1], "r-", "kf-filter")
```



Backward: Smoothing

<img src='img/LDS_smooth.svg', width = 110, height=100>

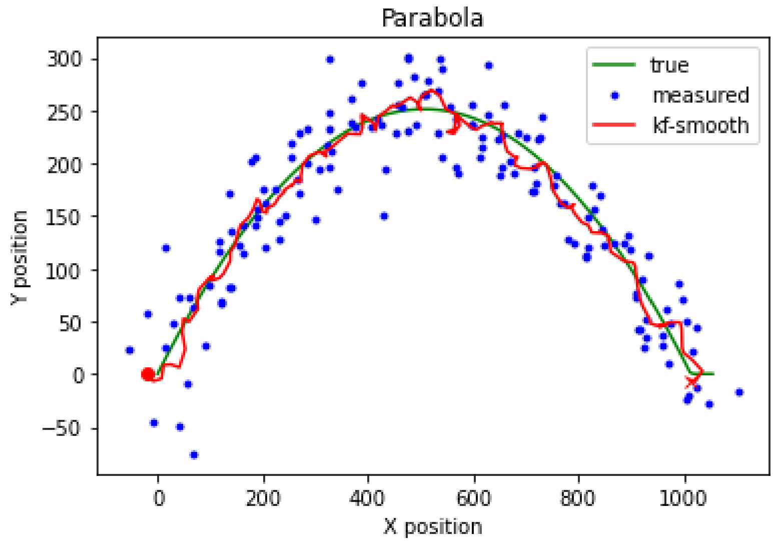
obtain estimates by propagating from x_n back to x_1 using results of forward pass $(\mu_n^{innov}, V_n^{innov}, V_n^{pred})$

$$N(z_n|\mu_n^{smooth}, V_n^{smooth})$$
$$\mu_n^{smooth} = \mu_n^{innov} + J_n(\mu_{n+1}^{smooth} - A\mu_n^{innov})$$
$$V_n^{smooth} = V_n^{innov} + J_n(V_{n+1}^{smooth} - V_{n+1}^{pred})J_n^T$$
$$J_N = V_n^{innov}A^T(V_{n+1}^{pred})^{-1}$$

This gives us the final estimate for z_n.

$$\hat{\mu}_n = \mu_n^{smooth}$$
$$\hat{V}_n = V_n^{smooth}$$

In [5]:
```
# Kalman smoothing
smoothed_state_means, smoothed_state_covariances = kf.smooth(data)
fig = plot_kalman(x,y,nx,ny, smoothed_state_means[:,0], smoothed_state_m
eans[:,1], "r-", "kf-smooth")
```



Kalman Filter Implementation

In this part of the exercise, you will implement the Kalman filter. Specifically, you need to implement the following method:

- filter: assume learned parameters, perform the forward calculation
- smooth: assume learned parameters, perform both the forward and backward calculation

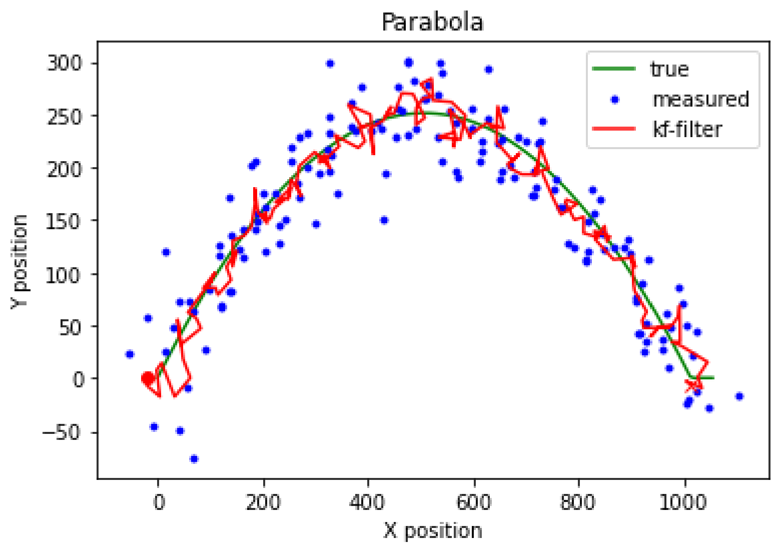
In [6]:
```
class MyKalmanFilter:
    """
    Class that implements the Kalman Filter
    """
    def __init__(self, n_dim_state=2, n_dim_obs=2):
        """
        @param n_dim_state: dimension of the laten variables
        @param n_dim_obs: dimension of the observed variables
        """
        self.n_dim_state = n_dim_state
        self.n_dim_obs = n_dim_obs
        self.transition_matrices = np.eye(n_dim_state)
        self.transition_offsets = np.zeros(n_dim_state) # you can ignore
this one, not used
        self.transition_covariance = np.eye(n_dim_state)
        self.observation_matrices = np.eye(n_dim_obs, n_dim_state)
        self.observation_covariance = np.eye(n_dim_obs)
        self.observation_offsets = np.zeros(n_dim_obs) # you can ignore
 this one, not used
        self.initial_state_mean = np.zeros(n_dim_state)
        self.initial_state_covariance = np.eye(n_dim_state)


    def filter(self, X):
        """
        Method that performs Kalman filtering
        @param X: a numpy 2D array whose dimension is [n_example, self.n
_dim_obs]
        @output: filtered_state_means: a numpy 2D array whose dimension
 is [n_example, self.n_dim_state]
        @output: filtered_state_covariances: a numpy 3D array whose dime
nsion is [n_example, self.n_dim_state, self.n_dim_state]
        """

        # validate inputs
        n_example, observed_dim = X.shape
        assert observed_dim==self.n_dim_obs

        # create holders for outputs
        filtered_state_means = np.zeros( (n_example, self.n_dim_state) )
        filtered_state_covariances = np.zeros( (n_example, self.n_dim_st
ate, self.n_dim_state) )

        #############################
        # TODO: implement filtering #
        #############################

        filtered_state_means[0] = self.initial_state_mean
        filtered_state_covariances[0] = self.initial_state_covariance

        for i in range(0, n_example):
            if i == 0:
                mean_0 = self.initial_state_mean
                sigma_0 = self.initial_state_covariance

            else:
```

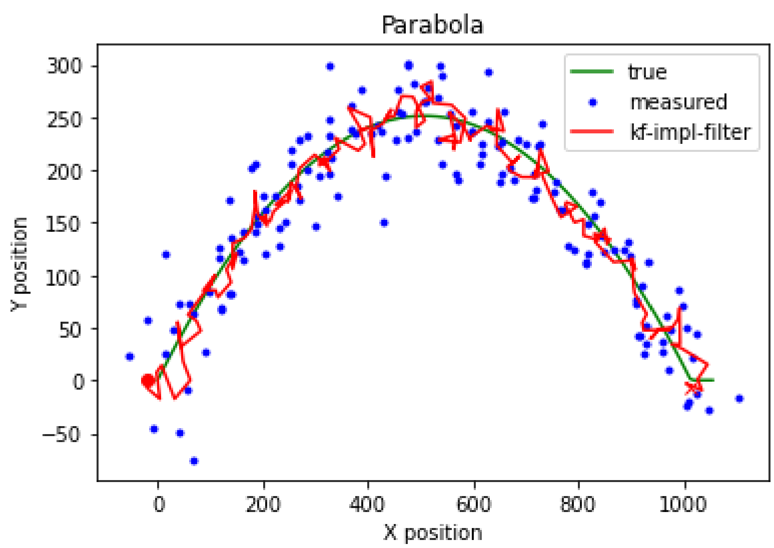
```
                mean_1_0 = self.transition_matrices @ mean_0
                sigma_1_0 = (self.transition_matrices @ sigma_0 @ self.t
ransition_matrices.T) + self.transition_covariance

                kalman_gain = sigma_1_0 @ self.observation_matrices.T @
np.linalg.inv((self.observation_matrices @ sigma_1_0 @ self.observation_
matrices.T) + self.observation_covariance)

                mean_0 = mean_1_0 + (kalman_gain @ (X[i] - (self.observa
tion_matrices @ mean_1_0)))
                sigma_0 = sigma_1_0 - (kalman_gain @ self.observation_ma
trices @ sigma_1_0)

                filtered_state_means[i, :] = mean_0
                filtered_state_covariances[i, :, :] = sigma_0

        return filtered_state_means, filtered_state_covariances

    def smooth(self, X):
        """
        Method that performs the Kalman Smoothing
        @param X: a numpy 2D array whose dimension is [n_example, self.n
_dim_obs]
        @output: smoothed_state_means: a numpy 2D array whose dimension
 is [n_example, self.n_dim_state]
        @output: smoothed_state_covariances: a numpy 3D array whose dime
nsion is [n_example, self.n_dim_state, self.n_dim_state]
        """
        # TODO: implement smoothing

        # validate inputs
        n_example, observed_dim = X.shape
        assert observed_dim==self.n_dim_obs

        # run the forward path
        mu_list, v_list = self.filter(X)

        # create holders for outputs
        smoothed_state_means = np.zeros( (n_example, self.n_dim_state) )
        smoothed_state_covariances = np.zeros( (n_example, self.n_dim_st
ate, self.n_dim_state) )

        #############################
        # TODO: implement smoothing #
        #############################

        smoothed_state_means[-1] = mu_list[-1]
        smoothed_state_covariances[-1] = v_list[-1]

        for i in range(n_example - 2, -1, -1):

            cov_i_i = v_list[i]
            cov_iplus1_i = (self.transition_matrices @ v_list[i] @ self.
transition_matrices.T) + self.transition_covariance

            F_i = cov_i_i @ self.transition_matrices.T @ np.linalg.inv(c
```

```
ov_iplus1_i)

            smoothed_state_means[i] = mu_list[i] + (F_i @ (smoothed_stat
e_means[i+1, :] - (self.transition_matrices @ mu_list[i])))
            smoothed_state_covariances[i] = (F_i @ (smoothed_state_covar
iances[i+1] - cov_iplus1_i) @ F_i.T) + v_list[i]

        return smoothed_state_means, smoothed_state_covariances

    def import_param(self, kf_model):
        """
        Method that copies parameters from a trained Kalman Model
        @param kf_model: a Pykalman object
        """
        need_params = ['transition_matrices', 'observation_matrices', 't
ransition_offsets',
                      'observation_offsets', 'transition_covariance',
                      'observation_covariance', 'initial_state_mean', 'initi
al_state_covariance']
        for param in need_params:
            setattr(self, param, getattr(kf_model, param))
```

Filtering

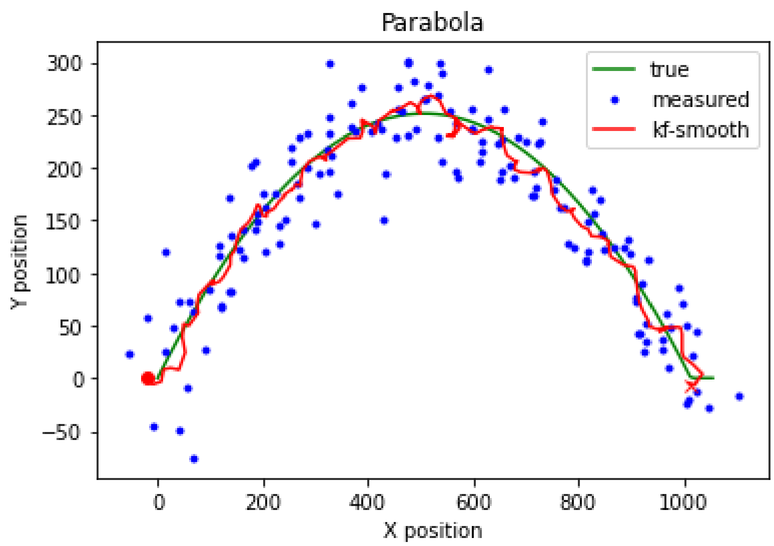
In [7]:
```
kf = KalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1])
kf.em(data)
my_kf = MyKalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1
])
my_kf.import_param(kf)
filtered_state_means, filtered_state_covariances = kf.filter(data)
filtered_state_means_impl, filtered_state_covariances_impl = my_kf.filte
r(data)
_ = plot_kalman(x,y,nx,ny, filtered_state_means[:,0], filtered_state_mea
ns[:,1], "r-", "kf-filter")
_ = plot_kalman(x,y,nx,ny, filtered_state_means_impl[:,0], filtered_stat
e_means_impl[:,1], "r-", "kf-impl-filter")
```

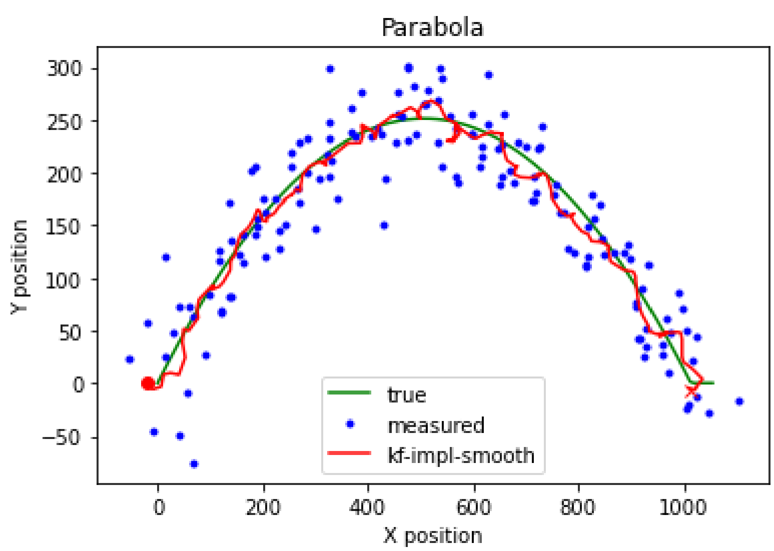




Smoothing

In [8]:
```
kf = KalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1])
kf.em(data)
my_kf = MyKalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1
])
my_kf.import_param(kf)
smoothed_state_means, smoothed_state_covariances = kf.smooth(data)
smoothed_state_means_impl, smoothed_state_covariances_impl = my_kf.smoot
h(data)
fig = plot_kalman(x,y,nx,ny, smoothed_state_means[:,0], smoothed_state_m
eans[:,1], "r-", "kf-smooth")
fig = plot_kalman(x,y,nx,ny, smoothed_state_means_impl[:,0], smoothed_st
ate_means_impl[:,1], "r-", "kf-impl-smooth")
```





Please turn in the code before 10/06/2020 11:59 pm EST. Please name your notebook netid.ipynb.

Your work will be evaluated based on the code and plots. You don't need to write down your answers to these questions in the text blocks.

In [ ]:

```

```