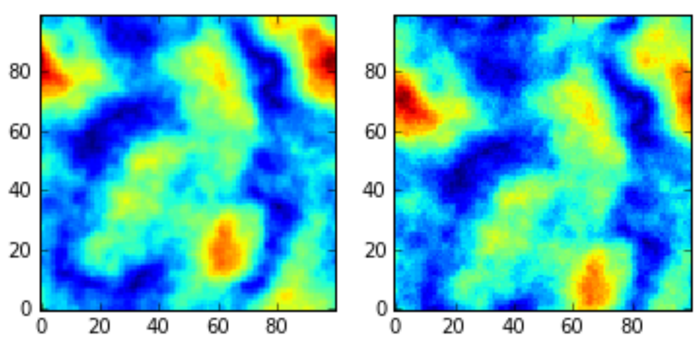
The Cross-Correlation package is available on github: https://github.com/keflavich/image_registration.

The goal is to determine the offset between two images with primarily extended structure.

In [1]:
```
# import statement (with warnings silenced).
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    import image_registration
errmsgs = np.seterr(all='ignore') # silence warning messages about div-by-zero
```

```
Activating auto-logging. Current session state plus future input saved.
Filename       : /Volumes/disk4/gbt/AGBT12B_221_01/ipython_log_2012-09-08.py
Mode           : append
Output logging : True
Raw input log  : False
Timestamping   : False
State          : active
 Logging to /Volumes/disk4/gbt/AGBT12B_221_01/ipython_log_2012-09-08.py
```

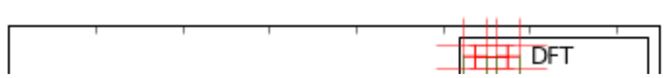
In [2]:
```
# create a simulated image by randomly sampling from a power-law power spectrum with
im1 = image_registration.tests.make_extended(100)
# create an offset version corrupted by noise
im2 = image_registration.tests.make_offset_extended(im1, 4.76666, -12.333333333333333
subplot(121); img1=imshow(im1)
subplot(122); img2=imshow(im2)
```

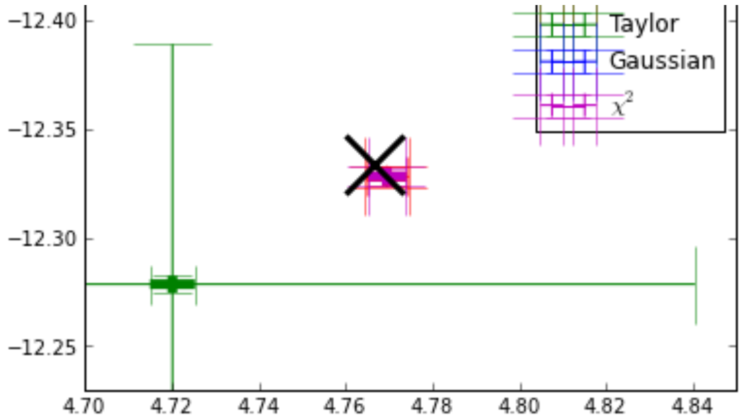


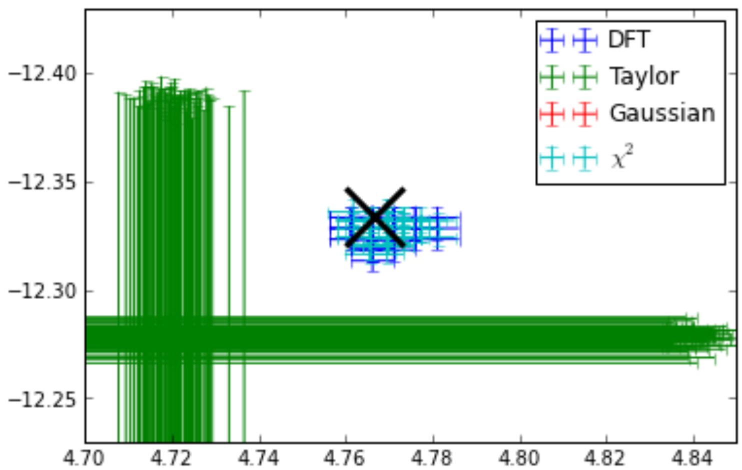
In [3]:
```
# Run the registration methods 100 times each (and hide the output)
offsets_n1,eoffsets_n1 = image_registration.tests.compare_methods(im1,im2,noise=0.1)
```

In [4]:
```
# plot the simulation data
# (note that the "gaussian" approach is hidden; it was problematic)
image_registration.tests.plot_compare_methods(offsets_n1,eoffsets_n1,dx=4.76666666,dy
figure(2); ax=axis([4.7,4.85,-12.23,-12.43])
figure(1); ax=axis([4.7,4.85,-12.23,-12.43])
# the outputs below show the x,y standard deviations (i.e., the "simulated error"),
# the means of the reported errors (i.e., the measured errors)
# and the ratio of the measured error to the simulated error - should be ~1 if correc
# the black X is the correct answer
```

```
Standard Deviations:  [ 0.00456276  0.00438376  0.00516853  0.00389744  0.
0.
  0.00429528  0.00413325]
Error Means:  [ 0.00497512  0.00497512  0.12037047  0.11054405  0.          0.
  0.00423828  0.0046875 ]
emeans/stds:  [  1.09037575   1.13489906  23.28909224  28.36321925          nan
          nan   0.98673067   1.13409595]
```



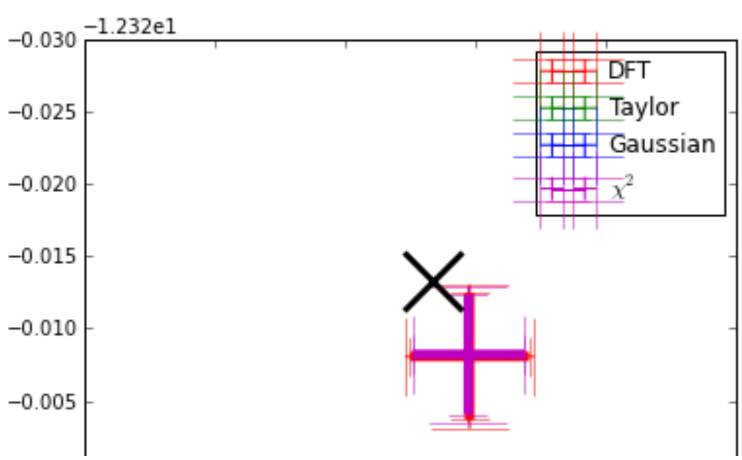


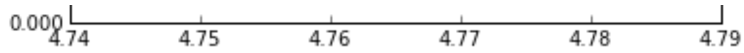


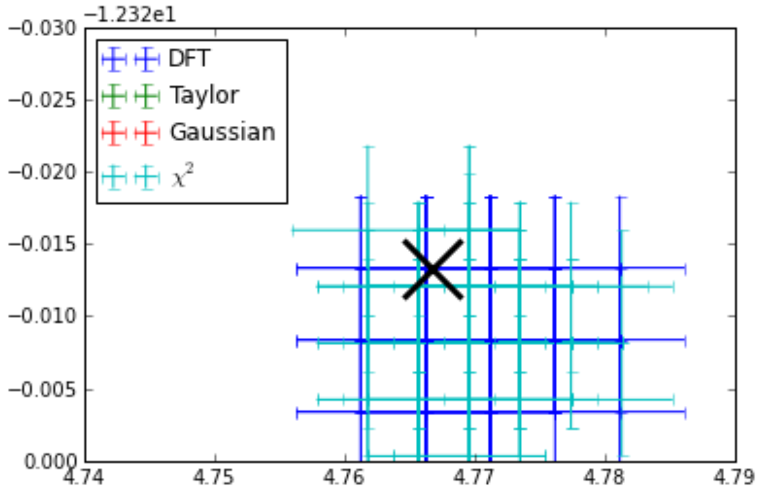
In [5]:

```
# plot the simulation data but zoomed in more (same as above otherwise)
# (note that the "gaussian" approach is hidden; it was problematic)
image_registration.tests.plot_compare_methods(offsets_n1,eoffsets_n1,dx=4.76666666,dy
figure(2); ax=axis([4.74,4.79,-12.32,-12.35])
figure(1); ax=axis([4.74,4.79,-12.32,-12.35])
# the outputs below show the x,y standard deviations (i.e., the "simulated error"),
# the means of the reported errors (i.e., the measured errors)
# and the ratio of the measured error to the simulated error - should be ~1 if correc
# the black X is the correct answer
```

```
Standard Deviations:  [ 0.00456276  0.00438376  0.00516853  0.00389744  0.
0.
  0.00429528  0.00413325]
Error Means:  [ 0.00497512  0.00497512  0.12037047  0.11054405  0.          0.
  0.00423828  0.0046875 ]
emeans/stds:  [  1.09037575   1.13489906  23.28909224  28.36321925          nan
          nan   0.98673067   1.13409595]
```







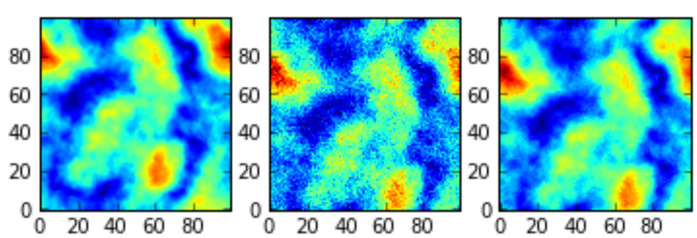
So how do these methods work? They all use the peak of the cross-correlation, which is most efficiently done via fourier transforms, to determine the offset.

The "cross_correlation_shift" function selects the cross-correlation peak, then finds the sub-pixel shift using a second order Taylor expansion.

The "register_images" function uses some linear algebra + fourier space tricks to upsample the image to determine sub-pixel shifts.

The "chi2_shift" function uses the same trick, but "automatically" determines the upsampling factor based on the $\Delta\chi^2$ values. The peak is identified, as is a region within 1σ (for 2 fitted parameters, $\Delta\chi^2 < 2.3$, then the original image is magnified to include only the 1σ region.
The errors are determined by marginalizing over the other fitted parameter, BUT it is possible to return the full $\Delta\chi^2$ image if you are concerned with correlation.

In [6]:
```
# create a simulated image by randomly sampling from a power-law power spectrum with
# don't re-make random image... im1 = image_registration.tests.make_extended(100)
# create an offset version corrupted by noise
im2noisy = image_registration.tests.make_offset_extended(im1, 4.76666, -12.3333333333
subplot(131); img1=imshow(im1)
subplot(132); img2=imshow(im2noisy)
subplot(133); img2=imshow(im2)
```

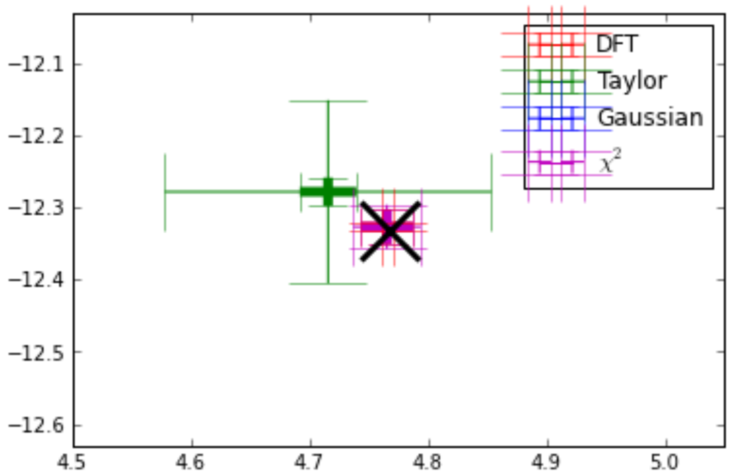


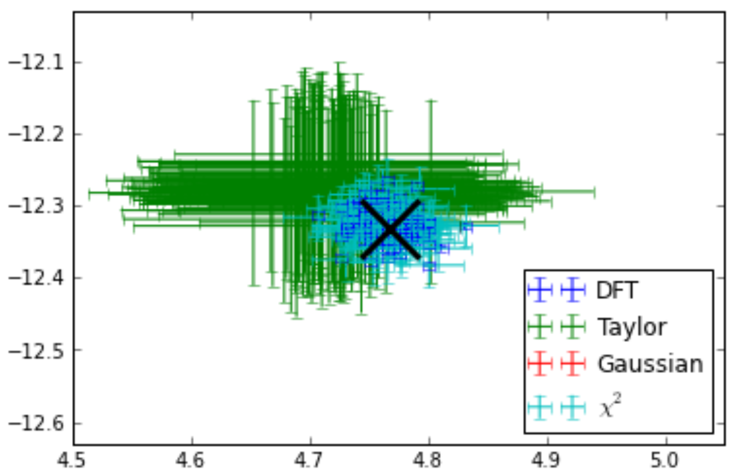
In [7]:
```
# Run the registration methods 100 times each (and hide the output)
offsets_n5,eoffsets_n5 = image_registration.tests.compare_methods(im1,im2,noise=0.5)
```

In [8]:
```
# plot the simulation data
# (note that the "gaussian" approach is hidden; it was problematic)
image_registration.tests.plot_compare_methods(offsets_n5,eoffsets_n5,dx=4.76666666,dy
figure(2); ax=axis([4.5,5.05,-12.63,-12.03])
figure(1); ax=axis([4.5,5.05,-12.63,-12.03])
# the outputs below show the x,y standard deviations (i.e., the "simulated error"),
```

```
# the means of the reported errors (i.e., the measured errors)
# and the ratio of the measured error to the simulated error - should be ~1 if correc
# the black X is the correct answer
```

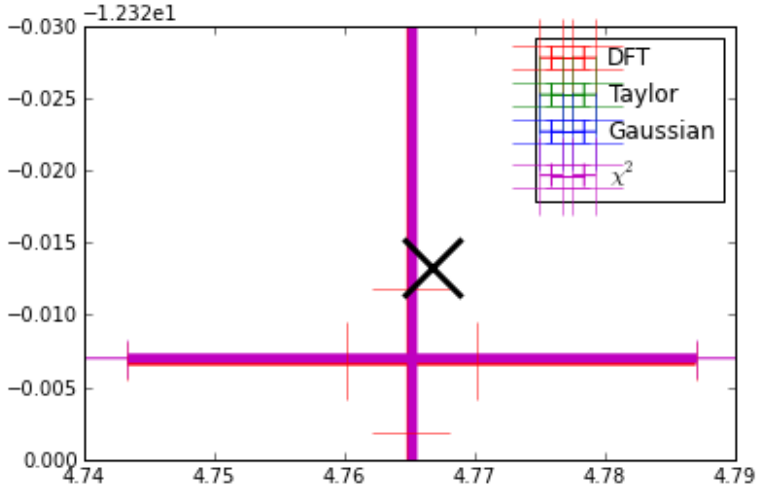
```
Standard Deviations:  [ 0.02184051  0.02353286  0.0238039   0.01956646  0.
0.
  0.02186998  0.02339381]
Error Means:  [ 0.00497512  0.00497512  0.13799409  0.12679361  0.          0.
  0.02845703  0.03056641]
emeans/stds:  [ 0.22779339  0.21141177  5.79711994  6.48015051         nan
nan
  1.30119163  1.30660248]
```

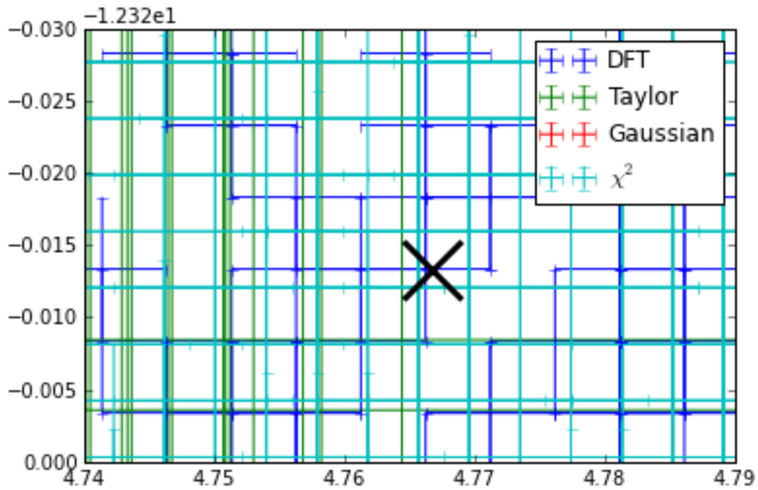




In [9]:
```
# plot the simulation data but zoomed in more (same as above otherwise)
# (note that the "gaussian" approach is hidden; it was problematic)
image_registration.tests.plot_compare_methods(offsets_n5,eoffsets_n5,dx=4.76666666,dy
figure(2); ax=axis([4.74,4.79,-12.32,-12.35])
figure(1); ax=axis([4.74,4.79,-12.32,-12.35])
# the outputs below show the x,y standard deviations (i.e., the "simulated error"),
# the means of the reported errors (i.e., the measured errors)
# and the ratio of the measured error to the simulated error - should be ~1 if correc
# the black X is the correct answer
```

```
Standard Deviations:  [ 0.02184051  0.02353286  0.0238039   0.01956646  0.
0.
  0.02186998  0.02339381]
Error Means:  [ 0.00497512  0.00497512  0.13799409  0.12679361  0.          0.
  0.02845703  0.03056641]
emeans/stds:  [ 0.22779339  0.21141177  5.79711994  6.48015051         nan
nan
  1.30119163  1.30660248]
```


−1.232e1
−0.030
−0.025
−0.020
−0.015
−0.010
−0.005
0.000
4.74
4.75
4.76
4.77
4.78
4.79
DFT
Taylor
Gaussian
χ^2


−1.232e1
−0.030
−0.025
−0.020
−0.015
−0.010
−0.005
0.000
4.74
4.75
4.76
4.77
4.78
4.79
DFT
Taylor
Gaussian
χ^2

In [9]: