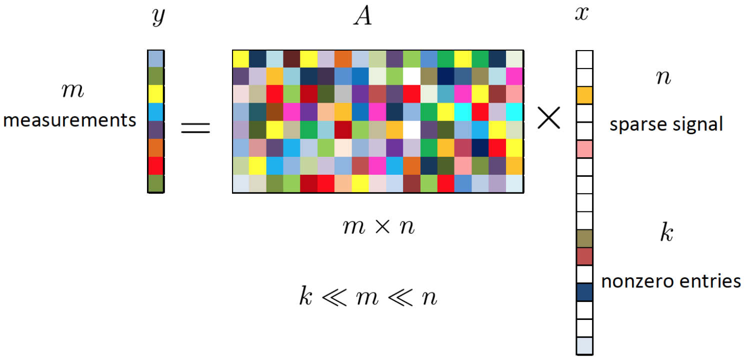
Picture of the Measurement Process



- *Measurement* process visualized

Recovering Sparse Solution

- In principle we can find the sparsest solution by minimizing of the zero-norm
- $||x||_0$ is the number of non-zero entires in x

$$\min ||x||_0, \text{ such that, } y = Ax \tag{1}$$

Greedy or Matching Pursuit

- Greedy: make the naive best choice at every iteration and build the estimate of x recursively
- *Matching Pursuit* algorithm
  - Select best dictionary element (d_i, y)
  - Update the coefficient and determine new residual

Orthogonal Matching Pursuit

- The matching pursuit algorithm was originally designed (Mallat and Zhang) to fit overcomplete dictionaries of time-frequency atoms to a signal

- for $k = 1, 2, 3, \ldots$

$$r^k = y - A\hat{x}^{k-1} \tag{2}$$

$$j^* = \arg\min_{j,x} ||r^k - a_j x||_2 \tag{3}$$

- Add j^* to the support and recalculate the coefficients

Coding using the imp Class in Python

- Code listing below shows the creation of a test vector in python and the syntax for decomposing it using the omp class
- Note that we specify in advance the number of non-zero coefficients, we define the sparsity

```
# Generate a signal
y = np.linspace(0, resolution - 1, resolution)
first_quarter = y < resolution / 4
y[first_quarter] = 3.
y[np.logical_not(first_quarter)] = -1.

omp = OrthogonalMatchingPursuit(n_nonzero_coefs=10)
omp.fit(np.transpose(D_multi), y)
coef = omp.coef_
idx_r, = coef.nonzero()
plt.plot(coef)

plt.show()
```

In-Class Exercise

- Use the code provided on the moodle to perform matching pursuit analysis on a test vector
- The output of your program should be a plot that compares the original test vector versus it's reconstruction for different levels of approximation (different sparsity, error)