→ **toyexample.py**

```
import matplotlib.pyplot as plt
from colgibbs import *
from stdgibbs import *


def _plot_joint_and_preds(filename, title, logjoints, logpreds):
    """Plots the log-joint probabilities and log-pred probabilites in a single window
       and saves the plot as 'filename'."""
    fig = plt.figure()
    plt.suptitle(title)

    ax1 = fig.add_subplot(211)
    ax1.set_ylabel('Log-joint prob')
    ax1.plot(logjoints)

    ax2 = fig.add_subplot(212)
    ax2.set_xlabel('Gibbs iteration')
    ax2.set_ylabel('Log-pred prob')
    ax2.plot(logpreds)

    plt.savefig(filename)


if __name__ == '__main__':
    data = np.loadtxt('toyexample.data', dtype=np.int)

    # standard Gibbs
    logjoints, logpreds, _, _, _, _, _, _ = std_gibbs(data, 3)
    _plot_joint_and_preds('std_lda_gibbs.png', 'standard LDA Gibbs', logjoints, logpreds)

    # collapsed Gibbs
    logjoints, logpreds, _, _, _, _ = col_gibbs(data, 3)
    _plot_joint_and_preds('collapsed_lda_gibbs.png', 'collapsed LDA Gibbs', logjoints, logpreds)
```

→ **stdgibbs.py**

```
"""Standard (i.e. uncollapsed) Gibbs sampler for LDA."""
import numpy as np
from numpy.random import dirichlet
from scipy.special import gamma
from utils import randinit_z, calc_counts


def _std_lda_logjoint(alpha, beta, A, B, theta, phi):
    """Calculates the log-joint of a standard (i.e. uncollapsed) LDA."""
    (D, K), W = A.shape, B.shape[1]

    logjoint = 0
    logjoint += K*np.log(gamma(W*beta))
```

```python
    logjoint += -W*K*np.log(gamma(beta))
    logjoint += D*np.log(gamma(K*alpha))
    logjoint += -K*D*np.log(gamma(alpha))
    logjoint += ((A + alpha - 1)*np.log(theta)).sum()
    logjoint += ((B + beta - 1)*np.log(phi)).sum()
    return logjoint


def _std_lda_logpred(Z, theta, phi, testc):
    """Calculate the log-predictive probs for standard LDA."""
    p = np.array([np.dot(theta[d], phi[:, w]) for d in Z for w in Z[d]])
    return np.dot(testc, np.log(p))


def std_gibbs(data, K, alpha=1, beta=1, nb_iters=200):
    """Gibbs sampler for the standard (i.e. uncollapsed) LDA."""
    Z = randinit_z(data, K)
    A, B, M = calc_counts(Z, K, max(data[:, 1]))
    theta = np.array([dirichlet(doc_c + alpha) for doc_c in A])
    phi = np.array([dirichlet(topic_c + beta) for topic_c in B])

    logjoints, logpreds = [], []
    logjoints.append(_std_lda_logjoint(alpha, beta, A, B, theta, phi))
    logpreds.append(_std_lda_logpred(Z, theta, phi, data[:, -1]))

    for i in range(nb_iters):
        for d in Z:
            for w in Z[d]:
                for wi, old_k in enumerate(Z[d][w]):
                    A[d, old_k] -= 1; B[old_k, w] -= 1; M[old_k] -= 1
                    probs = np.zeros(K)
                    for k in range(K):
                        probs[k] = phi[k, w] * theta[d, k]
                    probs /= np.sum(probs)

                    new_k = np.random.choice(len(probs), p=probs)
                    Z[d][w][wi] = new_k
                    A[d, new_k] += 1; B[new_k, w] += 1; M[new_k] += 1

            theta[d] = dirichlet(A[d] + alpha)

        phi = np.array([dirichlet(topic_c + beta) for topic_c in B])

        logjoints.append(_std_lda_logjoint(alpha, beta, A, B, theta, phi))
        logpreds.append(_std_lda_logpred(Z, theta, phi, data[:, -1]))

    return logjoints, logpreds, Z, theta, phi, A, B, M
```

→ **colgibbs.py**

```
"""Collapsed Gibbs sampler for LDA."""
import numpy as np
from scipy.special import gamma
from utils import randinit_z, calc_counts


def _col_lda_logjoint(alpha, beta, A, B, M):
    """Calculates the log-joint of a collapsed LDA."""
    (D, K), W = A.shape, B.shape[1]
    N = np.sum(A, axis=1)  # array of size D, d-th elem holds the total number of words in document
d

    logjoint = 0
    logjoint += D*np.log(gamma(K*alpha))
    logjoint += -K*D*np.log(gamma(alpha))
    logjoint += K*np.log(gamma(W*beta))
    logjoint += -W*K*np.log(gamma(beta))
    logjoint += np.log(gamma(A + alpha)).sum() - np.log(gamma(N + K*alpha)).sum()
    logjoint += np.log(gamma(B + beta)).sum() - np.log(gamma(M + W*beta)).sum()
    return logjoint


def _col_lda_logpred(Z, alpha, beta, A, B, M, testc):
    """Calculates the log-preditive probs for collapsed LDA."""
    Pdk = alpha + A
    Pdk = Pdk / Pdk.sum(axis=1)[:, np.newaxis]

    Pkw = beta + B
    Pkw = Pkw / Pkw.sum(axis=1)[:, np.newaxis]

    p = np.array([np.dot(Pdk[d], Pkw[:, w]) for d in Z for w in Z[d]])
    return np.dot(testc, np.log(p))

def col_gibbs(data, K, alpha=1, beta=1, nb_iters=200):
    """Gibbs sampler for the collapsed LDA."""
    Z = randinit_z(data, K)
    W = max(data[:, 1])
    A, B, M = calc_counts(Z, K, W)

    logjoints, logpreds = [], []
    logjoints.append(_col_lda_logjoint(alpha, beta, A, B, M))
    logpreds.append(_col_lda_logpred(Z, alpha, beta, A, B, M, data[:, -1]))

    for i in range(nb_iters):
        for d in Z:
            for w in Z[d]:
                for wi, old_k in enumerate(Z[d][w]):
                    A[d, old_k] -= 1; B[old_k, w] -= 1; M[old_k] -= 1

                    probs = np.zeros(K)
```

```
            for k in range(K):
                probs[k] = (A[d][k] + alpha) * ((B[k,w] + beta) / (M[k] + W*beta))
            probs /= np.sum(probs)


            new_k = np.random.choice(len(probs), p=probs)
            Z[d][w][wi] = new_k
            A[d, new_k] += 1; B[new_k, w] += 1; M[new_k] += 1


        logjoints.append(_col_lda_logjoint(alpha, beta, A, B, M))
        logpreds.append(_col_lda_logpred(Z, alpha, beta, A, B, M, data[:, -1]))

    return logjoints, logpreds, Z, A, B, M
```

→ **utils.py**

```
"""Helper functions that are common for both Gibbs samplers for LDA."""
from collections import defaultdict
import numpy as np


def randinit_z(data, K):
    """Randomly initialize Z from a uniform distribution. Z is represented as
    a dictionary, where Z[d][w] is a numpy array where the i-th element contains
    the topic corresponding to the i-th instance of the word w in document d."""
    Z = defaultdict(lambda: defaultdict(np.array))
    for d, w, train_count, _ in data:
        Z[d-1][w-1] = np.random.choice(K, train_count)
    return Z


def calc_counts(Z, K, W):
    """Calculates the counts A, B & M from topic indicators Z."""
    A = np.zeros((len(Z), K), dtype=np.int)  # element (d,k) holds the number of words in document
d assigned to topic k
    B = np.zeros((K, W), dtype=np.int)  # element (k, w) holds the number of times word w is
assigned to topic k across all documents
    M = np.zeros(K, dtype=np.int)  # element k holds the total number of words assigned to topic k
across all documents

    for k in range(K):
        for d in Z:
            for w in Z[d]:
                total = sum(Z[d][w] == k)
                A[d, k] += total
                B[k, w] += total
                M[k] += total

    return A, B, M
```