TR-IT-0185

Implementation and Testing of Quasi-Bayes Speaker Adaptation Algorithm

ディミトリ・リティシェフ
Dimitry Rtischev

シャン・ホウ
Qiang Huo

ハラルド・シンガー
Harald Singer

1996.09

The Quasi-Bayes speaker adaptation algorithm enables incremental adaptation of HMM parameters as speech from the target speaker becomes available, without requiring that the adaptation speech be stored for subsequent adaptation. The algorithm has been implemented as part of the ATR SSS-ToolKit recognition system and evaluated on the ATR spontaneous speech database. Experimental results indicate that the Quasi-Bayes adaptation algorithm is a computationally efficient method for achieving consistent improvement in recognition accuracy in supervised mode, i.e., relying on manually prepared transcriptions of the adaptation speech. The results also show that the algorithm does not yield consistent improvement in recognition accuracy in unsupervised mode. Additional research is necessary to achieve unsupervised adaptation capability.

目次

1 Introduction

Speaker independent models of speech are trained on a diverse set of speakers and typically yield good recognition performance for most speakers. However, recognition accuracy for a particular speaker can be improved by adapting speaker independent models to the speech of that speaker. This report discusses the implementation and performance evaluation of the Quasi-Bayes speaker adaptation algorithm. A key advantage of this algorithm is its ability to incrementally adapt hidden Markov model (HMM) parameters as speech from the target speaker becomes available, without requiring that the adaptation speech be stored for subsequent adaptation. The theory underlying the algorithm is discussed in [4], [3], and [2]. This report describes the implementation of this algorithm as part of the SSS-ToolKit speech recognition system [9]. This report also documents experiments conducted to evaluate the performance of the algorithm using the ATR spontaneous speech database [6].

Experimental results presented below indicate that the Quasi-Bayes adaptation algorithm is a computationally efficient method for achieving consistent improvement in recognition accuracy in supervised mode, i.e., relying on manually prepared transcriptions of the adaptation speech. The results also show that the algorithm does not yield consistent improvement in recognition accuracy in unsupervised mode. Additional research is necessary to achieve unsupervised adaptation capability.

2 Implementation

2.1 Program Usage

`Exe.QBadapt_HMnet` is an executable program that inputs HMM models, adaptation hyperparameters, and adaptation speech data, and produces adapted HMM models and an updated set of hyperparameters. `QBadapt_HMnet` reads adaptation speech data from standard input in the `.sssdata` format. Filenames of speaker independent models (input), speaker adapted models (output) and adaptation hyperparameters (output) must be specified on the command line.

The command line usage is:

```
Exe.QBadapt_HMnet -if file [-ih file] -of file -oh file [-it n][-tt n][-pw n][-fc n]
        [-rw n][-cm text][-dm file][-vb n] < adaptation-speech.sssdata
```

The command line options are:

-cm (string) Comment string.

-dm (string) Disk memory file name.

-vb (int) Verbose mode flag. Default is 0 (quiet).

-it (int) Number of training iterations. Default is 1.

-tt (int) Training type: Forward-backward is selected by `-tt 0`; segmental (Viterbi) training is selected by `-tt n`, where `n` is a positive integer which determines how often a new Viterbi alignment is generated. For example, `-tt 2` generates a new Viterbi alignment once every two training iterations. Default is 0 (forward-backward).

-pw (float) Prior weight is used in computing initial hyperparameters. This parameter controls the relative importance of the speaker-independent training data and speaker-dependent adaptation data. Higher values of prior weight increase the contribution of the speaker-independent training data. Must be positive. Default is 4.0.

-fc (float) Exponential forgetting coefficient. This parameter controls how fast past adaptation data is forgotten. In other words, it conrols the relative importance of the latest adaptation data. Smaller values correspond to faster forgetting. Must be in the range [0,1]. Default is 1.0 (no forgetting).

-rw (float) Refresh Weight. This parameter is used only in case of parameter refreshing, which is performed in order to reduce the contribution of training and adaptation speech observed so far relative to the contribution of subsequent adapation speech. Smaller values correspond to less contribution of past speech. Must be in the range [0.001,1]. Default is 1.0.

-if (string) Input HMnet filename. Must be specified. If filename suffix is .bin, binary format is assumed.

-ih (string) Input hyperparameter filename. If not specified, initial hyperparameters are computed automatically.

-of (string) Output HMnet filename. Must be specified. If filename suffix is .bin, binary format is used.

-oh (string) Output hyperparameter filename. Must be specified.

2.2 Source code structure

The Quasi-Bayes adaptation algorithm has been implemented for the case of hidden Markov models with output probability distributions being mixtures of Gaussians with diagonal covariance matrices. The adaptation algorithm is implemented in the following C source code files:

`main_QBadapt_HMnet.c` This file contains `main()` and routines for reading command line options and checking their validity.

`QBadapt_HMnet.c` This file serves as an interface between the top-level program `main_QBadapt_HMnet.c` and basic computation routines in `forward_backward.c` and `QBhyper_math.c`

`QBhyper_math.c` This file implements the main computations of algorithm, namely the updating of hyperparameters and reestimation of HMM parameters.

`QBhyper_util.c` This file contains utility routines including memory allocation, freeing, and file input/output.

`QBadapt.h` This file defines data structures and provides function prototypes.

Furthermore, the implementation of the Quasi-Bayes algorithm relies on the forward-backward and Viterbi algorithms which are implemented in the file `forward_backward.c` as part of the SSS-ToolKit package.

2.3 Data structures

The parameters used by the Quasi-Bayes adaptation algorithm are called "hyperparameters" and are stored in the following data structures:

HYPERPARAM Top-level hyperparameter data structure:

```
typedef struct {
  int num_states;                /* total number of states in HMNet */
  int dimension;                 /* dimensionality of gaussians */
  STATE_HYPERS *state_hypers;    /* hyperparameters for each state */
} HYPERPARAM;
```

STATE_HYPERS Hyperparameter data structure for a single state:

```
typedef float t_real;  /* facilitates switching between double and float */

typedef struct {
  int num_gaussians;   /* number of gaussians in the mixture for this state */
  t_real *eta;         /* hyperparameters for state transition matrix */
  t_real *nu;          /* hyperparameters for mixture weights */
  t_real *tau;         /* hyperparameters for gaussian means */
  t_real **mu;         /* hyperparameters for gaussian means */
  t_real *alpha;       /* hyperparameters for gaussian cov matrix */
  t_real **beta;       /* hyperparameters for gaussian cov matrix */
} STATE_HYPERS;
```

2.4 Algorithm implementation

The following indices are used consistently in formulas and program code:

```
i, j: indicate which state
k: indicates which Gaussian distribution within a mixture in a given state
d: indicates which dimension within a Gaussian distribution
```

The function `QBadapt_HMnet()` is the entry point to the adaptation algorithm:

```
void QBadapt_HMnet (int data_num,              /* number of data samples */
                    DATA **data,               /* adaptation speech data */
                    HMNET *cdhmm,              /* HMM to be adapted (input and output)*/
                    HYPERPARAM *old_hypers,    /* current hyperparameters (input) */
                    HYPERPARAM *new_hypers,    /* new hyperparameters (output) */
                    int num_iters,             /* number of iterations */
                    float forget_coeff,        /* rho (forgetting coefficient) */
                    int training_type)         /* nonzero if Viterbi training */
```

`QBadapt_HMnet()` iterates over the given speech data and performs the three main stages of the adaptation algorithm:

Stage 1 One iteration of the forward-backward or segmental (Viterbi) training is performed on the adaptation speech data. The purpose is to compute following sufficient statistics for reestimating hyperparameters and HMM parameters:

$$g_{ij} = \sum_{t=1}^{T_n} \gamma_t(i,j) \tag{1}$$

$$c_{ik} = \sum_{t=1}^{T_n} \zeta_t(i,k) \tag{2}$$

$$M_{ikd} = \sum_{t=1}^{T_n} \zeta_t(i,k) x_{td} \tag{3}$$

$$V_{ikd} = \sum_{t=1}^{T_n} \zeta_t(i,k) x_{td}^2 \tag{4}$$

where

$$\gamma_t(i,j) = \Pr(s_t = i, s_{t+1} = j | \mathbf{X}, \lambda) \tag{5}$$

is the probability of state i at time t and state j at time $t+1$, or, equivalently, probability of the transition from state i to state j being taken at time t (given adaptation speech data $\mathbf{X}$ and current HMM parameters λ); and

$$\zeta_t(i,k) = \Pr(s_t = i, l_t = k | \mathbf{X}, \lambda) \tag{6}$$

is the probability of observation x_t in mixture component k of state i.

Stage 2 The function `update_hypers()` computes new hyperparameters from previous hyperparameters, HMM parameters, and counts computed by forward-backward or segmental (Viterbi) training.

```
void update_hypers (HYPERPARAM *old_hypers,  /* (n) hyperparameters (input) */
                    HYPERPARAM *new_hypers,  /* (n+1) hyperparameters (output) */
                    float forget_coeff,      /* forgetting coefficient rho */
                    HMNET *cdhmm,            /* HMM to be adapted */
                    double **zeta_counts,    /* Count2_Buf in forward-backward.c */
                    double **gamma_counts,   /* Count3_Buf in forward-backward.c */
                    double **mean_counts,    /* Mean_Buf in forward-backward.c */
                    double **var_counts)     /* Var_Buf in forward-backward.c */
```

New hyperparameters are computed according to the following formulas:

$$\eta_{ij}^{(n+1)} = \rho \cdot (\eta_{ij}^{(n)} - 1) + 1 + g_{ij} \tag{7}$$

$$\nu_{ik}^{(n+1)} = \rho \cdot (\nu_{ik}^{(n)} - 1) + 1 + c_{ik} \tag{8}$$

$$\tau_{ik}^{(n+1)} = \rho \tau_{ik}^{(n)} + c_{ik} \tag{9}$$

$$\mu_{ikd}^{(n+1)} = \frac{\rho\tau_{ik}^{(n)}\mu_{ikd}^{(n)} + c_{ik}\bar{x}_{ikd}}{\rho\tau_{ik}^{(n)} + c_{ik}} \tag{10}$$

$$\alpha_{ik}^{(n+1)} = \rho\cdot(\alpha_{ik}^{(n)} - 0.5) + 0.5 + \frac{1}{2}c_{ik} \tag{11}$$

$$\beta_{ikd}^{(n+1)} = \rho\beta_{ikd}^{(n)} + \frac{1}{2}S_{ikd} + \frac{\rho\tau_{ik}^{(n)}c_{ik}}{2(\rho\tau_{ik}^{(n)} + c_{ik})}(\bar{x}_{ikd} - \mu_{ikd}^{(n)})^2 \tag{12}$$

where:

$$\bar{x}_{ikd} = \frac{M_{ikd}}{c_{ik}} = \frac{\sum_{t=1}^{T_n}\zeta_t(i,k)x_{td}}{\sum_{t=1}^{T_n}\zeta_t(i,k)} \tag{13}$$

$$S_{ikd} = V_{ikd} - \frac{M_{ikd}^2}{c_{ik}} = \sum_{t=1}^{T_n}\zeta_t(i,k)(x_{td} - \bar{x}_{ikd})^2 \tag{14}$$

Stage 3 HMM parameters are updated based on the newly computed hyperparameters.

The updating of HMM parameters from hyperparameters is implemented in

```
void update_models_from_hypers (HMNET *cdhmm, HYPERPARAM *h)
```

according to the following formulas:

1. Transition probabilities:

$$\hat{a}_{ij} = \frac{\eta_{ij}^{(n+1)} - 1}{\sum_{j=1}^{N}\eta_{ij}^{(n+1)} - N} \tag{15}$$

2. Mixture weights:

$$\hat{\omega}_{ik} = \frac{\nu_{ik}^{(n+1)} - 1}{\sum_{k=1}^{K}\nu_{ik}^{(n+1)} - K} \tag{16}$$

3. Gaussian mean vectors:

$$\hat{m}_{ikd} = \mu_{ikd}^{(n+1)} \tag{17}$$

4. Gaussian covariance matrices:

$$\hat{\sigma}_{ikd}^2 = \frac{\beta_{ikd}^{(n+1)}}{\alpha_{ik}^{(n+1)} - 0.5} \tag{18}$$

2.5 Hyperparameter initialization

If an input hyperparameter file is provided to the Exe.QBadapt_HMnet program via the *-ih* command line option, those hyperparameters are used as the initial hyperparameters. In this case, the prior weight *-pw* command line option is ignored.

If an input hyperparameter file is not specified to Exe.QBadapt_HMnet, then the prior weight *-pw* value is used to initialize hyperparameters. The initialization is performed by the function

```
void initialize_hypers (HMNET *cdhmm,      /* models */
                        HYPERPARAM *h,     /* hyperparameters */
                        float tau)         /* prior weight */
```

according to the following formulas,

$$\eta_{ij}^{(0)} = 1 + \tau_{ik}^{(0)} \tag{19}$$

$$\nu_{ik}^{(0)} = 1 + \tau_{ik}^{(0)} \tag{20}$$

$$\mu_{ik}^{(0)} = m_{ik}^{(SI)} \tag{21}$$

$$\alpha_{ik}^{(0)} = \frac{1}{2} + \frac{1}{2}\tau_{ik}^{(0)} \tag{22}$$

$$\beta_{ikd}^{(0)} = \frac{1}{2}\tau_{ik}^{(0)}\cdot\sigma_{ikd}^{2(SI)} \tag{23}$$

where the hyperparameters for Gaussian means $\tau_{ik}^{(0)}$ are initialized using the prior weight τ supplied via the *-pw* command line option:

$$\tau_{ik}^{(0)} = \tau \tag{24}$$

It should be noted that a more informative method for initializing $\tau_{ik}^{(0)}$ is

$$\tau_{ik}^{(0)} = \omega_{ik} \sum\nolimits_k \tau \tag{25}$$

as discussed in [5].

Alternatively, initial hyperparameters can be computed at the conclusion of speaker independent training. The following function can be called after the last iteration of SI training in order to compute initial hyperparameters and save them to a file:

```
void init_hypers_from_si_counts (HMNET *cdhmm,          /* models */
                                 HYPERPARAM *h,         /* hyperparameters */
                                 float prior_wt,        /* prior weight (epsilon) */
                                 double **zeta_counts,  /* Count2_Buf in forward-backward.c */
                                 double **gamma_counts, /* Count3_Buf in forward-backward.c */
                                 int *state_counts)     /* number of times each state occured
                                                           in SI training data */
```

Prior weight is a parameter which controls the relative importance of the speaker-independent training data and speaker-dependent adaptation data. Setting the prior weight to 1 maximizes the contribution of the speaker-independent training data. The prior weight supplied by the user is normalized for each state in proportion to the amount of training data the state receives (N_i is the minimum number of times state i was traversed during speaker-independent training). The initial hyperparameters are computed from forward-backward counts corresponding to the speaker-independent (SI) training data according to the following formulas:

$$\eta_{ij}^{(0)} = 1 + \frac{\epsilon_i}{N_i} \cdot g_{ij}^{(SI)} \tag{26}$$

$$\nu_{ik}^{(0)} = 1 + \frac{\epsilon_i}{N_i} \cdot c_{ik}^{(SI)} \tag{27}$$

$$\tau_{ik}^{(0)} = \frac{\epsilon_i}{N_i} \cdot c_{ik}^{(SI)} \tag{28}$$

$$\mu_{ik}^{(0)} = m_{ik}^{(SI)} \tag{29}$$

$$\alpha_{ik}^{(0)} = \frac{1}{2} + \frac{1}{2}\frac{\epsilon_i}{N_i} \cdot c_{ik}^{(SI)} \tag{30}$$

$$\beta_{ikd}^{(0)} = \frac{1}{2}\frac{\epsilon_i}{N_i} \cdot \sigma_{ikd}^{2(SI)} c_{ik}^{(SI)} \tag{31}$$

where ϵ is the prior weight.

Experiments have shown that initial hyperparameters generated from SI training counts do not yield good adaptation performance. This is potentially due to the inadequacy of the normalization scheme. The simpler initialization scheme implemented in `initialize_hypers()` yields much better performance. Consequently, all experiments described below have been performed using initial hyperparameters computed via `initialize_hypers()`.

2.6 Hyperparameter refreshing

Refreshing hyperparameters reduces their contribution in subsequent adapatation. This is useful in cases when the speaker or channel conditions suddenly change, or when it is desired to reduce the contribution of past adaptation data relative to the contribution of subsequent adaptation data. Hyperparameters are refreshed by the following function:

```
void refresh_hypers (HYPERPARAM *h,  /* hyperparameters to be refreshed */
                     float rw)       /* refresh weight (theta) */
```

Refresh weight ($0.001 < \theta \leq 1.0$) is a weighting coefficient that controls the reduction of contribution of current hyperparameters to subsequent adaptation. Reducing the value of θ reduces the contribution of the prior knowledge in subsequent adaptation.

Hyperparameters are refreshed according to the following formulas:

$$\hat{\eta}_{ij} = 1 + \theta \cdot (\eta_{ij} - 1) \quad (32)$$

$$\hat{\nu}_{ik} = 1 + \theta \cdot (\nu_{ik} - 1) \quad (33)$$

$$\hat{\tau}_{ik} = \theta \cdot \tau_{ik} \quad (34)$$

$$\hat{\mu}_{ik} = \mu_{ik} \quad (35)$$

$$\hat{\alpha}_{ik} = 0.5 + \theta \cdot (\alpha_{ik} - 0.5) \quad (36)$$

$$\hat{\beta}_{ikd} = \theta \cdot \beta_{ikd} \quad (37)$$

2.7 File input/output

Hyperparameter file input/output is performed by the following functions in QBhypers_util.c:

```
void read_hypers(char *fn,  HYPERPARAM *h)
void write_hypers(char *fn, HYPERPARAM *h)
```

Hyperparameter files are binary. Platform-independence is achieved by using conditional compilation flags

```
#if MACHINE_BYTEORDER != NETWORK_BYTEORDER
```

to determine when bit swapping is necessary. If necessary, bit swapping is performed by functions in `ConvertPara.c`.

The hyperparameter file format is shown in Table 1. It consists of a two-integer header followed by hyperparameters for each state.

表 1: Hyperparameter file format (State index i ranges from 0 to HYPERPARAM.num_states-1)

Item type	Number of items	value
int	1	HYPERPARAM.dimension = DIMENSION
int	1	HYPERPARAM.num_states
int	1	HYPERPARAM.state_hypers[i].num_gaussians = GAUSS_NUM(i)
t_real	TRANS_NUM	HYPERPARAM.state_hypers[i].eta
t_real	GAUSS_NUM(i)	HYPERPARAM.state_hypers[i].nu
t_real	GAUSS_NUM(i)	HYPERPARAM.state_hypers[i].tau
t_real	GAUSS_NUM(i)	HYPERPARAM.state_hypers[i].alpha
t_real	DIMENSION	HYPERPARAM.state_hypers[i].mu[0]
...	...	...
t_real	DIMENSION	HYPERPARAM.state_hypers[i].mu[GAUSS_NUM(i)-1]
t_real	DIMENSION	HYPERPARAM.state_hypers[i].beta[0]
...	...	...
t_real	DIMENSION	HYPERPARAM.state_hypers[i].beta[GAUSS_NUM(i)-1]
...	...	...
...	...	...

3 Performance Evaluation: Experimental Design

3.1 Speech data

The female portion of the ATR Spontaneous Speech Database was used to evaluate the performance of the Quasi-Bayes adaptation algorithm [6].

Training data for speaker indepependent models consisted of 2,576 utterances (turns) from 197 conversations by 196 females.

Testing data was divided into the following sets:

- Batch adaptation set

```
FHITA-adapt-90
FYOAS-adapt-90
FYUYO-adapt-90
```

- Batch test set

```
FHITA-test-56
FYOAS-test-35
FYUYO-test-40
```

- Incremental adaptation/test set - consists of all testing speech available for each speaker.

```
FHITA-adapttest-146  (concatenation of FHITA-adapt-90 and FHITA-test-56)
FYOAS-adapttest-125  (concatenation of FYOAS-adapt-90 and FYOAS-test-35)
FYUYO-adapttest-130  (concatenation of FYUYO-adapt-90 and FYUYO-test-40)
FYOMA-adapttest-75
FHITO-adapttest-49
FYOOO-adapttest-20
FYUKI-adapttest-19
FYOAZ-adapttest-18
FMASZ-adapttest-18
FMAKA-adapttest-18
```

3.2 Baseline experiments

Three baseline experiments were carried out in order to establish benchmarks for evaluating the performance of the Quasi-Bayes adaptation algorithm.

Baseline 1 No adaptation: recognition with speaker-independent models

- Recognize batch test set with the initial models
- Recognize incremental adaptation/test set with the initial models

Baseline 2 VFS adaptation (batch, supervised)

- Adapt initial models on the batch adaptation set
- Recognize the batch test set with the adapted models
- Repeat with 1,5,10,20,50,90 adaptation utterances

Baseline 3 MAP adaptation (batch, supervised)

- Adapt initial models on the batch adaptation set
- Recognize the batch test set with the adapted models
- Repeat with 1,5,10,20,50,90 adaptation utterances

3.3 Supervised adaptation experiments

Supervised adaptation experiments were carried out to test the performance of the Quasi-Bayes adaptation algorithm when the correct transcription of adaptation speech is available.

Test 1a QB adaptation (batch, supervised) = MAP

- Adapt INITIAL models on the batch adaptation set
- Recognize the batch test set with the adapted models
- Repeat with 1,5,10,20,50,90 adaptation utterances

More precisely, the sequence of steps is:

1. first adapt INITIAL models on 1 sentence from adaptation set
2. then, recognize entire test set with resulting models
3. then, adapt INITIAL models on 5 sentences from the adaptation set
4. then, recognize entire test set with the resulting models
5. etc.

Test 1b QB adaptation (block-incremental, supervised)

- Adapt LATEST models on the next block of speech from the batch adaptation set
- Recognize the batch test set with the adapted models
- Use the following block sizes: 1,5,10,20,50,90 adaptation utterances

More precisely, the sequence of steps is:

1. first adapt INITIAL models on 1 sentence from adaptation set
2. then, recognize entire test set with the resulting (LATEST) models
3. then, adapt the LATEST models on 4 more sentences from the adaptation set
4. then, recognize entire test set with the resulting models
5. etc.

3.4 Unsupervised adaptation experiments

Unsupervised adaptation experiments were carried out to test the performance of the Quasi-Bayes adaptation algorithm when the correct transcription of adaptation speech is not available. Unsupervised adaptation relies on potentially inaccurate transcription of adaptation speech automatically generated by a recognizer.

Test 2a QB adaptation (batch, unsupervised)

- Recognize the batch adaptation set with INITIAL models
- Adapt INITIAL models on the batch adaptation set using the recognized result as the transcription string
- Recognize the batch test set with the ADAPTED models
- Repeat with 1,5,10,20,50,90 adaptation utterances

Test 2b QB adaptation (block-incremental, unsupervised)

- Recognize the batch adaptation set with INITIAL models
- Adapt LATEST models on the next block of speech from the batch adaptation set using the recognized result as the transcription string
- Recognize the batch test set with the adapted models
- Use the following block sizes: 1,5,10,20,50,90 adaptation utterances

3.5 Incremental adaptation experiments

Incremental adaptation experiments were carried out to simulate the condition of adaptation speech becoming available gradually over time.

Test 3a QB adaptation (incremental, supervised)

- Recognize utterance from incremental adaptation/test set with LATEST models
- Adapt models using the transcription, not the recognized result
- Repeat for all utterances in the adaptation/test set

Test 3b QB adaptation (incremental, unsupervised)

- Recognize utterance from incremental adaptation/test set with LATEST models
- Adapt models using the recognized result, not the transcription
- Repeat for all utterances in the adaptation/test set

Test 3c QB adaptation (incremental, unsupervised, self-learning)

The term *self-learning* means that several iterations of recognition and adaptation are performed on each utterence. Each iteration consists of a recognition step and an adaptation step. The recognition step generates a hypothesis which is used as the transcription in the adaptation step. The recognition result from the last iteration is used to measure performance accuracy. The models and hyperparameters output by the adaptation step in the last iteration are used as the initial models and hyperparameters for adapting on the next utterance. The models and hyperparameters output by the adaptation step in all but the last iteration are not used. More precisely, the sequence of steps is:

- Recognize an utterance from incremental adaptation/test set with LATEST models
- Adapt models using the recognized result and LASTEST models/hyperparameters, generating INTERIM models
- Re-recognize the same utterance with INTERIM models, record error rate
- Discard INTERIM models/hypers
- Adapt models using the LASTEST recognized result and LATEST models/hypers
- Update LATEST models/hypers
- Repeat for all utterances in the adaptation/test set

4 Performance Evaluation: Results

4.1 Comparison of Quasi-Bayes and VFS adaptation algorithms

The VFS adaptation algorithm is described in [1], [7], [8], and [10]. Supervised, batch-mode adaptation experiments carried out with three speakers show that VFS and Quasi-Bayes adaptation algorithms yield similar performance. It should be noted that batch-mode Quasi-Bayes algorithm is equivalent to MAP reestimation. In particular, the experiments have shown that VFS tends to perform better than QB/MAP for small amounts of adaptation speech, but asymptotically QB/MAP performs as well as or better than VFS. Table 2 and Figure 1 detail the phoneme recognition error rates as a function of the amount of adaptation speech input to VFS and QB/MAP algorithms. The experimental setup for QB adaptation was: 3 iterations with forward-backward training; initial hyperparameters generated with prior weight of 4. The experimental setup for VFS adaptation was: 3 iterations with smoothing rate of 10.

表 2: Quasi-Bayes and VFS Adapatation (Supervised, Batch, 3 iterations, FB training, Tau = 4, Smoothing rate = 10)

Speaker	Adapt Speech	Quasi-Bayes (Test1a)		VFS (Baseline2)	
	secs	%error	%impr	%error	%impr
FHITA	0	22.351	0.00	22.351	0.00
	4	22.738	-1.73	22.254	0.43
	14	19.787	11.47	19.981	10.60
	30	19.158	14.29	17.755	20.56
	49	18.191	18.61	16.884	24.46
	143	15.868	29.01	16.594	25.76
	267	14.949	33.12	15.239	31.82
FYOAS	0	16.567	0.00	16.567	0.00
	1	17.537	-5.86	16.791	-1.35
	11	17.687	-6.76	14.776	10.81
	16	18.060	-9.01	14.478	12.61
	43	15.597	5.855	14.179	14.41
	130	12.836	22.52	12.761	22.97
	271	11.194	32.43	10.970	33.78
FYUYO	0	20.398	0.00	20.398	0.00
	3	19.801	2.93	19.154	6.10
	20	19.005	6.83	18.010	11.71
	32	17.264	15.36	16.716	18.05
	57	16.418	19.51	16.368	19.76
	143	14.080	30.97	15.572	23.66
	253	14.080	30.97	15.721	22.93

4.2 Comparison of batch and incremental supervised adaptation

The Quasi-Bayes adaptation algorithm yields consistent improvement in recognition accuracy in supervised mode, i.e., relying on manually prepared transcriptions of the adaptation speech. Experiments show similar rates of accuracy improvement under batch, block-incremental, and incremental adaptation conditions. In batch mode, speaker independent models are used as the starting point for adaptation. In contrast, block-incremental adaptation uses models which were adapted on previous blocks of adaptation speech. In the limit, incremental adaptation adapts models after each conversation turn. Table 3 and Figure 2 detail

the phoneme recognition error rates on a separate test set as a function of the amount of adaptation speech input to the QB algorithm under batch, block-incremental, and incremental modes. The experimental setup was: single iteration with forward-backward training; initial hyperparameters generated with a prior weight of 4.

表 3: Supervised Quasi-Bayes Adaptation: Batch, Block-Incremental, Incremental (1 iteration, FB training, Tau = 4)

Speaker	Adapt Speech	Batch (Test1a)		Block-Incremental (Test1b)		Incremental (Test3a)	
	secs	%error	%impr	%error	%impr	%error	%impr
FHITA	0	22.351	0.00	22.351	0.00	22.351	0.00
	4	22.883	-2.38	22.883	-2.38	22.883	-2.38
	14	20.077	10.17	20.319	9.09	20.126	9.95
	30	18.820	15.80	19.400	13.20	19.545	12.55
	49	17.417	22.08	19.158	14.29	18.626	16.67
	143	15.723	29.65	17.755	20.56	16.642	25.54
	267	15.336	31.39	15.385	31.17	15.288	31.60
FYOAS	0	16.567	0.00	16.567	0.00	16.567	0.00
	1	18.134	-9.46	18.134	-9.46	18.134	-9.46
	11	17.836	-7.66	17.612	-6.31	17.612	-6.31
	16	17.388	-4.96	17.985	-8.56	17.239	-4.06
	43	16.418	0.90	17.090	-3.16	15.522	6.31
	130	12.612	23.87	12.463	24.77	13.284	19.82
	271	11.045	33.33	11.418	31.08	11.567	30.18
FYUYO	0	20.398	0.00	20.398	0.00	20.398	0.00
	3	19.950	2.20	19.950	2.20	19.950	2.20
	20	18.905	7.32	18.657	8.54	18.856	7.56
	32	17.264	15.36	17.363	14.88	17.861	12.44
	57	16.816	17.56	17.612	13.66	15.871	22.19
	143	14.179	30.49	15.821	22.44	14.577	28.54
	253	14.378	29.51	14.876	27.07	14.627	28.29

4.3 Comparison of incremental supervised and unsupervised adaptation

Ideally, incremental adaptation performance should be measured by adapting models on speech drawn from an adaptation data set and recognizing speech drawn from a separate test data set. However, testing on a separate data set is impractical because of the large data and computational requirements. Therefore, incremental adaptation performance has been measured using a single data set. The experimental setup involves recognizing an utterance, adapting models on that utterance, and then proceeding to recognize and adapt on the next utterance. A cumulative performance statistic is maintained to measure recognition accuracy as a function of the duration of adaptation speech. A single iteration with forward-backward learning was used. Initial hyperparameters were generatated using a prior weight of 4.

Unfortunately, this experimental setup does not permit direct measurement of adaptation performance. The difficulty stems from the fact that errors accumulated during the early stages of adaptation continue to influence cumulative performance statistics throughout the subsequent adaptation stages. To better illustrate the underlying learning behavior, Figures 3 through 12 show two views of the learning curve for every test speaker: cumulative percent error rate and cumulative number of errors. All errors (insertions, deletions, and substitutions) are counted equally.

The learning curves plotted in Figures 3 through 12 enable a comparison between speaker-independent, supervised speaker-adaptive, and unsupervised speaker-adaptive performance. The shape of the learning curves indicates that:

1. The Quasi-Bayes adaptation algorithm yields consistent improvement in recognition accuracy in supervised mode, i.e., relying on manually prepared transcriptions of the adaptation speech.

2. The Quasi-Bayes adaptation algorithm does not yield consistent improvement in recognition accuracy in unsupervised mode, i.e., relying on transcriptions of adaptation speech automatically generated by a recognizer.

5 Additional Experimental Findings

5.1 Initialization of hyperparameters

Experiments have shown that initial hyperparameters generated from SI training counts do not yield good adaptation performance. This is apparently due to the difficulty of properly normalizing for the different frequencies with which SI training tokens occur. Ideally, this normalization should be based on state occupancy counts derived via Viterbi alignment of SI training data. As an approximation, we instead used a count of triphone frequencies appearing in the transcriptions of SI training sentences. This corresponds to a lower bound on true occupancy counts derivable via Viterbi alignment. This approximation yielded initial hyperparameters which led to poor adaptation performance. However, it is arguable that even if Viterbi alignment were used to obtain true state occupancy counts, the difficulty with this hyperparameter initialization scheme would persist. Consistently normalizing parameters in a shared-parameter system is problematic.

The simpler initialization scheme implemented in `initialize_hypers()` yields much better performance. Consequently, all experiments described above have been performed using initial hyperparameters computed via `initialize_hypers()`, using prior weight $\tau = 4$.

5.2 Forward-backward and segmental training

The amount of computation required by the Quasi-Bayes adaptation algorithm can be reduced by using segmental training instead of forward-backward training. Segmental training is computationally more efficient than forward-backward training because it relies on a single time alignment of adaptation speech against a sequence of HMM states as generated by the Viterbi algorithm. Experiments show that segmental training is approximately 30% faster than forward-backward training and not significantly different in terms of recognition accuracy. For comparison purposes, Table 4 and Figure 13 show phonemene error rates observed with forward-backward and segmental training.

5.3 Number of EM training iterations

Experimental results show that the Quasi-Bayes adaptation algorithm performs well with only a single iteration of forward-backward or segmental training. Furthermore, experiments indicate that additional training iterations do not lead to improved performance. Consequently, the experiments described above have been performed with a single forward-backward or segmental training iteration. For comparison purposes, Table 5 and Figure 14 present phonemene error rates observed with one and three forward-backward iterations.

5.4 Computation time

Experiments show that the Quasi-Bayes adaptation algorithm is a practical algorithm from the standpoint of computational requirements. Tests carried out on an HP 735/125 workstation indicate that the algorithm requires time on the same order of magnitude as the duration of adaptation speech. The tests also show that the time consumed by the adaptation algorithm grows somewhat faster than linear as adaptation utterance duration increases. The most computationally-intensive part of the algorithm is the training stage: either forward-backward or segmental training. The evaluation of Gaussian densities by the function `cal_prob()` is the most time-consuming computation. Segmental training requires approximately two-thirds as much time as forward-backward training. Figure 15 shows the relationship between adaptation speech time and computation time for both forward-backward and segmental training modes.

5.5 Self-learning

Experiments show that self-learning has a minimal effect on adaptation performance. The reason is that the recognition result does not change betweeen self-learning iterations since each utterance does not contain

表 4: Effect of training methods: forward-backward versus segmental (Supervised, Incremental, 1 iteration, Tau = 4)

Speaker	Adapt Speech secs	Forward-Backward (Test3a) %error	%impr	Segmental (Test3a) %error	%impr
FHITA	0	22.351	0.00	22.351	0.00
	4	22.883	-2.38	22.835	-2.17
	14	20.126	9.95	20.464	8.44
	30	19.545	12.55	19.061	14.72
	49	18.626	16.67	18.578	16.88
	143	16.642	25.54	16.546	25.97
	267	15.288	31.60	15.143	32.25
FYOAS	0	16.567	0.00	16.567	0.00
	1	18.134	-9.46	17.985	-8.56
	11	17.612	-6.31	17.463	-5.41
	16	17.239	-4.06	17.239	-4.06
	43	15.522	6.31	15.597	5.86
	130	13.284	19.82	12.687	23.42
	271	11.567	30.18	11.493	30.63
FYUYO	0	20.398	0.00	20.398	0.00
	3	19.950	2.20	20.249	0.73
	20	18.856	7.56	18.756	8.05
	32	17.861	12.44	17.761	12.93
	57	15.871	22.19	16.368	19.76
	143	14.577	28.54	14.776	27.56
	253	14.627	28.29	14.776	27.56

enough data to effect a significant revision of model parameters. Only changes in the endpoints have been observed. Consequently, self-learning iterations do not have a significant effect on overall performance.

5.6 Hyperparameter data types

Experiments show that storing adapation hyperparameters as float rather than double does not lead to any degradation in adaptation performance. Since float is more memory-efficient, it is used by default.

表 5: Effect of additional forward-backward iterations (Supervised, Batch, Tau = 4)

Speaker	Adapt Speech secs	1 iteration (Test1a) %error	%impr	3 iterations (Test1a) %error	%impr
FHITA	0	22.351	0.00	22.351	0.00
	4	22.883	-2.38	22.738	-1.73
	14	20.077	10.17	19.787	11.47
	30	18.820	15.80	19.158	14.29
	49	17.417	22.08	18.191	18.61
	143	15.723	29.65	15.868	29.01
	267	15.336	31.39	14.949	33.12
FYOAS	0	16.567	0.00	16.567	0.00
	1	18.134	-9.46	17.537	-5.86
	11	17.836	-7.66	17.687	-6.76
	16	17.388	-4.96	18.060	-9.01
	43	16.418	0.90	15.597	5.855
	130	12.612	23.87	12.836	22.52
	271	11.045	33.33	11.194	32.43
FYUYO	0	20.398	0.0	20.398	0.00
	3	19.950	2.20	19.801	2.93
	20	18.905	7.32	19.005	6.83
	32	17.264	15.36	17.264	15.36
	57	16.816	17.56	16.418	19.51
	143	14.179	30.49	14.080	30.97
	253	14.378	29.51	14.080	30.97

6 Conclusion

The Quasi-Bayes speaker adaptation algorithm has been implemented as part of the SSS-ToolKit speech recognition system. Experiments have been conducted to evaluate the performance of the algorithm on spontaneous speech in the ATR spontaneous speech database. Experimental results indicate that the algorithm is a computationally efficient method for achieving consistent improvement in recognition accuracy when accurate transcriptions of adaptation speech are available. Additional research is necessary to develop methods capable of consistently improving recognition accuracy under unsupervised conditions.

参考文献

[1] H. Hattori and S. Sagayama. Vector field smoothing principle for speaker adaptation. In *Proceedings of International Conference on Spoken Language Processing*, pages 381–4, October 1992.

[2] Q. Huo and C.-H. Lee. On-line quasi-Bayes adaptation of CDHMM parameters for speech recognition. In *Proceedings of the Acoustical Society of Japan Fall 1995 Meeting*, pages 131–2, 1995.

[3] Q. Huo and C.-H. Lee. A study of on-line quasi-Bayes adaptation for CDHMM-based speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 705–8, May 1996.

[4] Q. Huo and C.H. Lee. On-line adaptive learning of the continuous density hidden Markov model based on approximate recursive Bayes estimate. *IEEE Transactions on Speech and Audio Processing*, 1996. (forthcoming).

[5] C.-H. Lee and J.-L. Gauvain. Speaker adaptation based on MAP estimation of HMM parameters. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages II–558–561, 1993.

[6] A. Nakamura, S. Matsunaga, T. Shimizu, M. Tonomura, and Y. Sagisaka. Japanese speech database for robust speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*, Philadelphia, October 1996. (forthcoming).

[7] K. Ohkura, M. Sugiyama, and S. Sagayama. Speaker adaptation based on transfer vector field smoothing with continuous mixture density HMMs. In *Proceedings of International Conference on Spoken Language Processing*, pages 369–72, October 1992.

[8] J. Takahashi and S. Sagayama. Vector-field-smoothed Bayesian learning for incremental speaker adaptation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages I–696–9, May 1995.

[9] J. Takami. SSS-toolkit(ver3.0) user's manual. Technical Report TR-IT-0039, ATR, 1995. (in Japanese).

[10] M. Tonomura, T. Kosaka, and S. Matsunaga. Speaker adaptation based on transfer vector field smoothing using maximum a posteriori probability estimation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages I–688–91, May 1995.

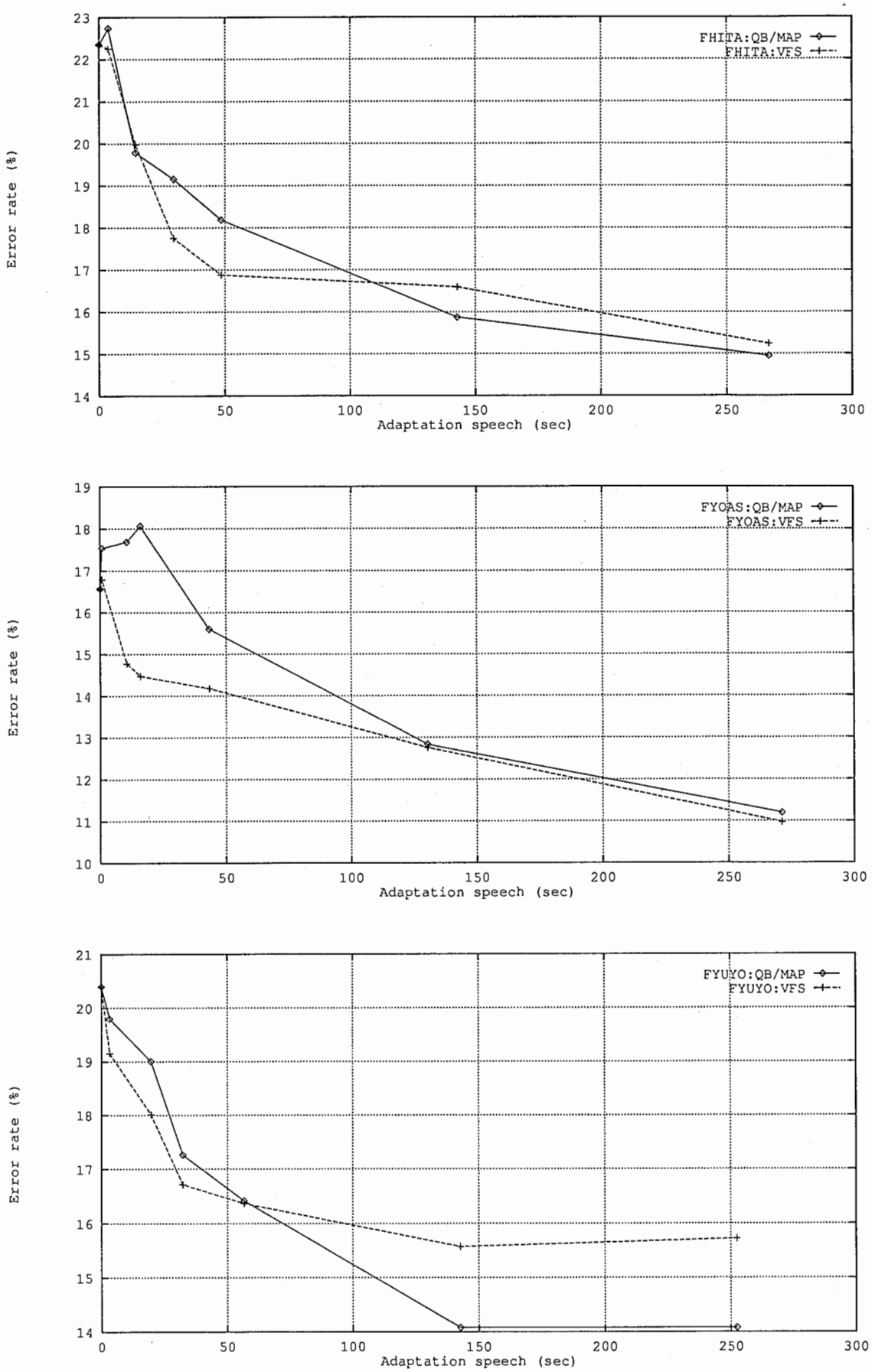


図 1: Quasi-Bayes and VFS Adapatation (Supervised, Batch, 3 iterations, FB training, Tau = 4, Smoothing rate = 10)

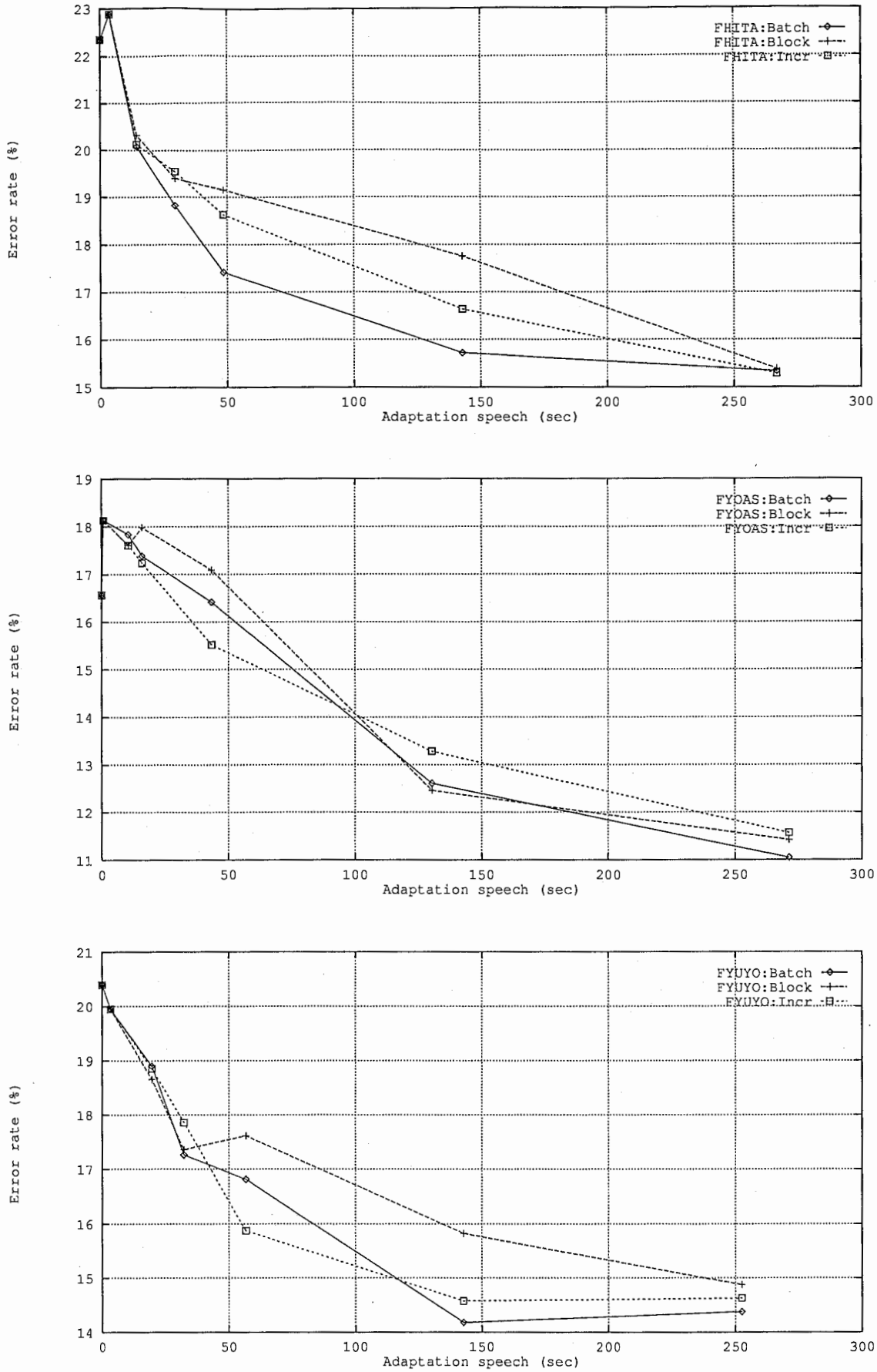


図 2: Supervised Quasi-Bayes Adaptation: Batch, Block-Incremental, Incremental (1 iteration, FB training, Tau = 4)

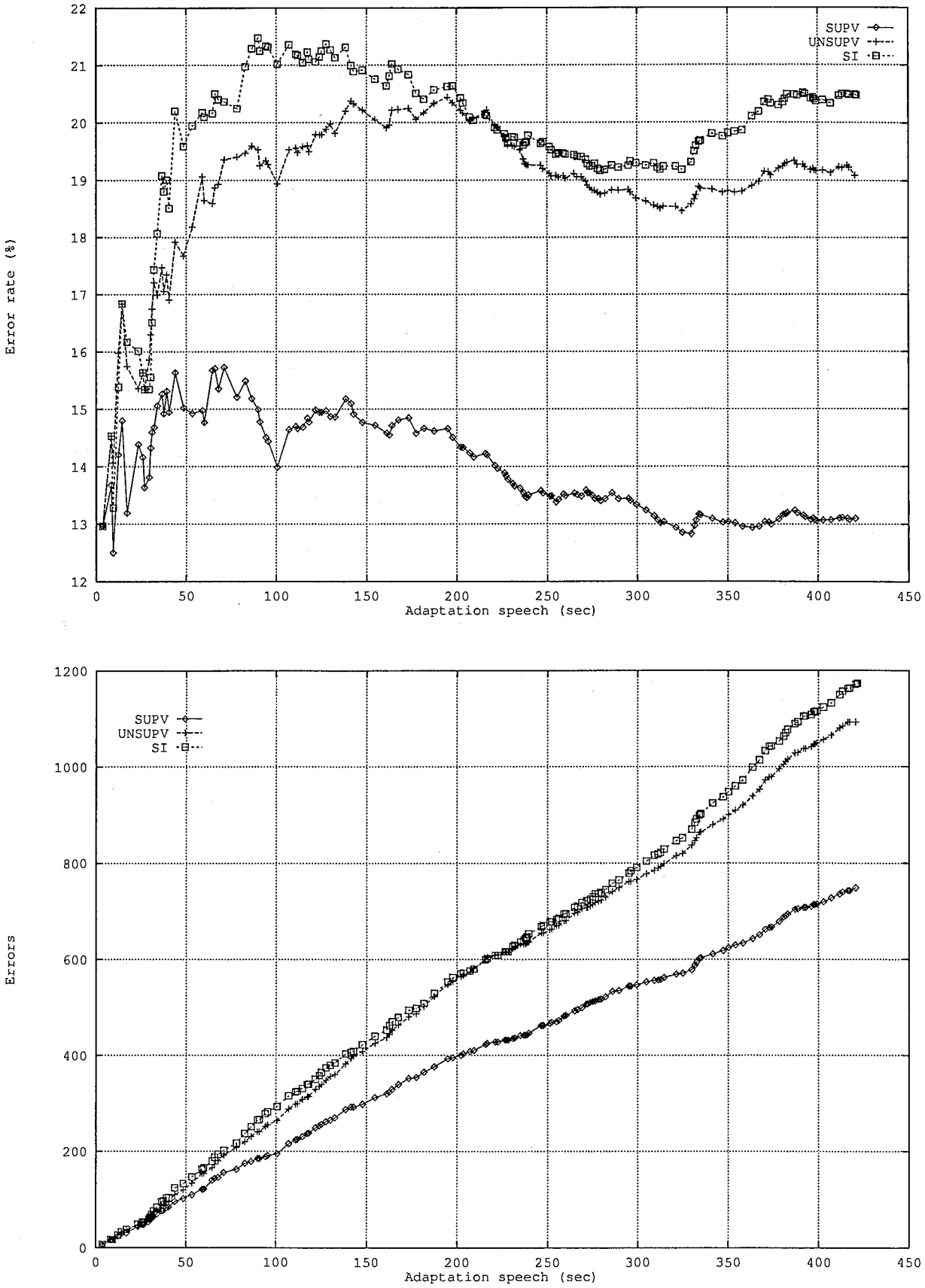
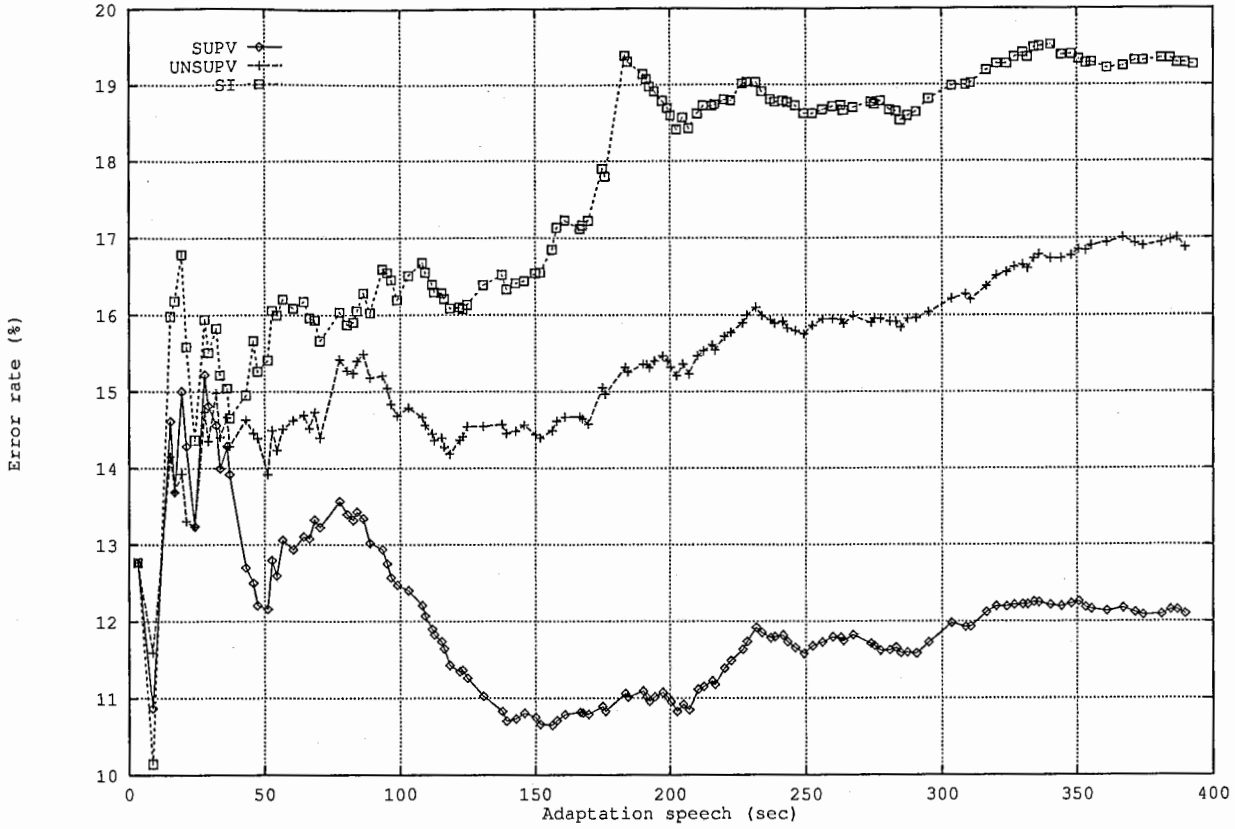


図 3: Incremental Quasi-Bayes Adaptation: Speaker FHITA



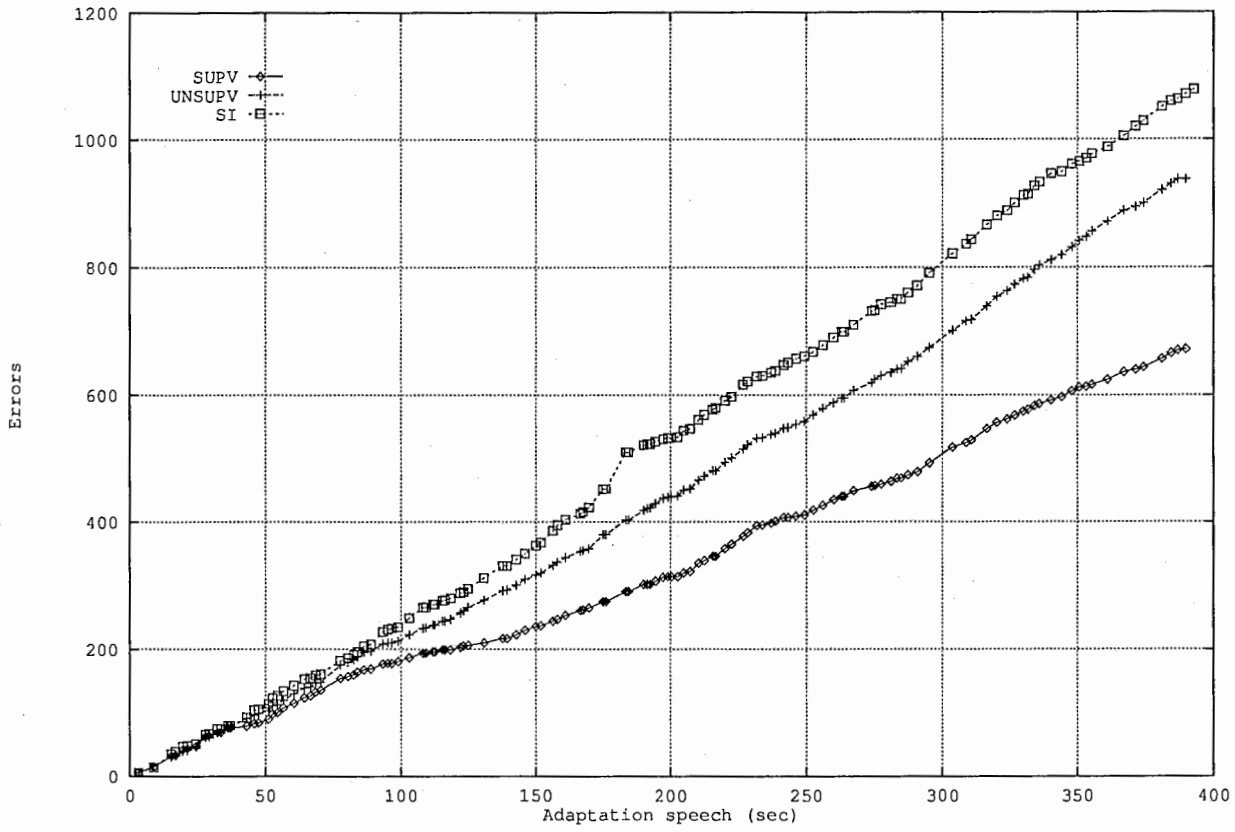
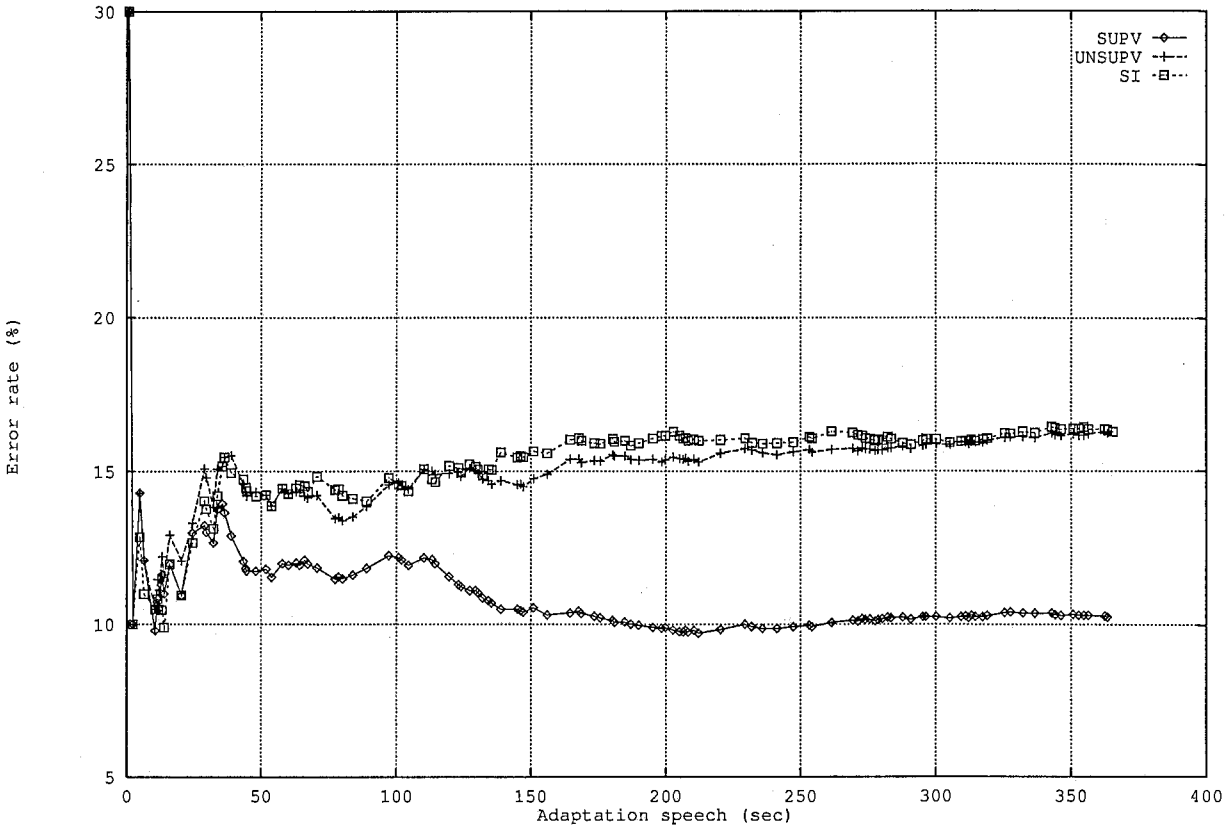


図 4: Incremental Quasi-Bayes Adaptation: Speaker FYUYO



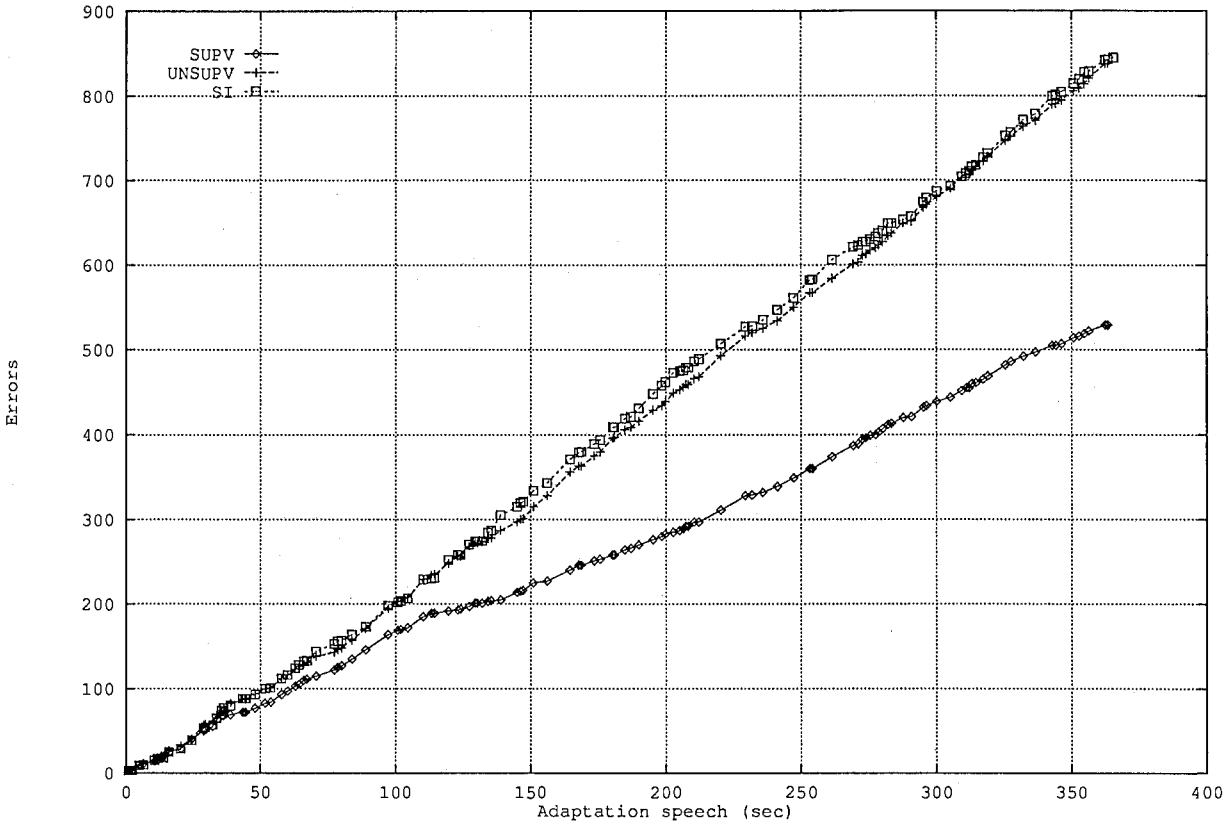
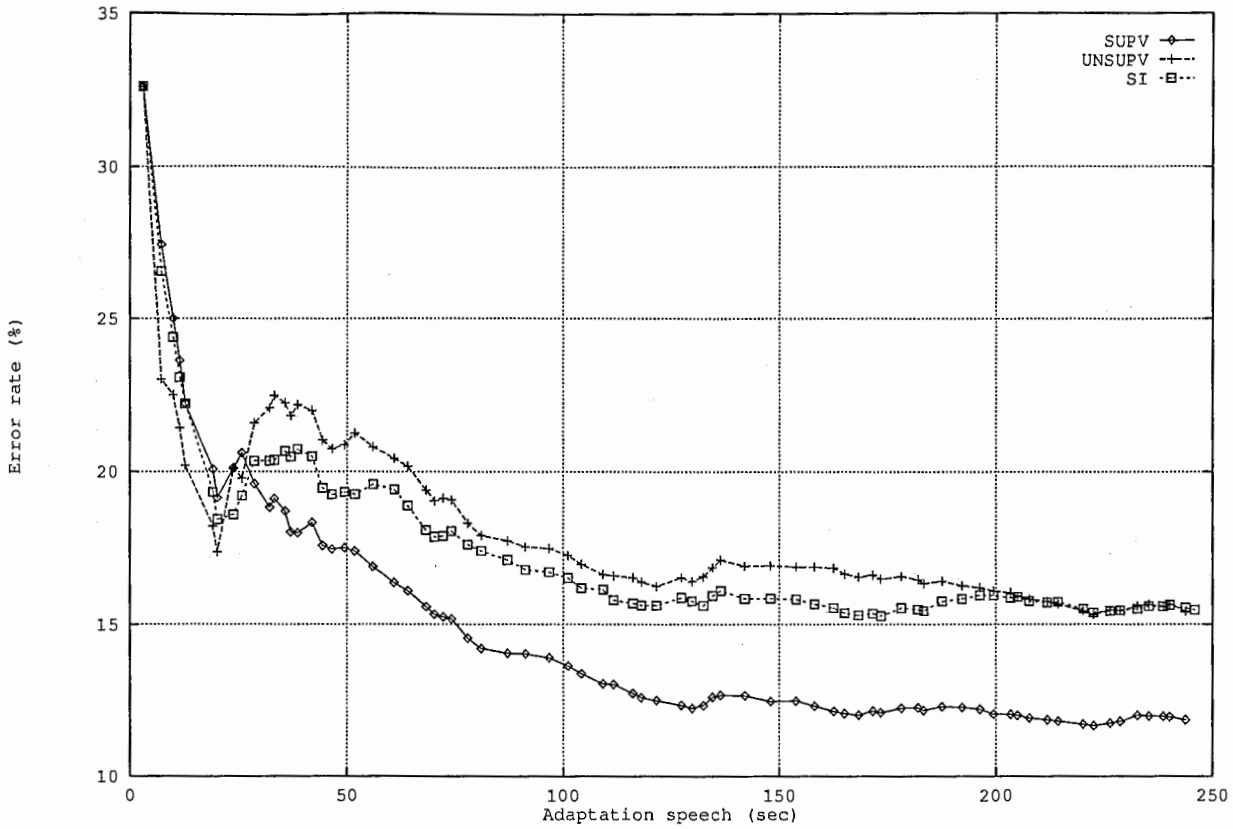


図 5: Incremental Quasi-Bayes Adaptation: Speaker FYOAS



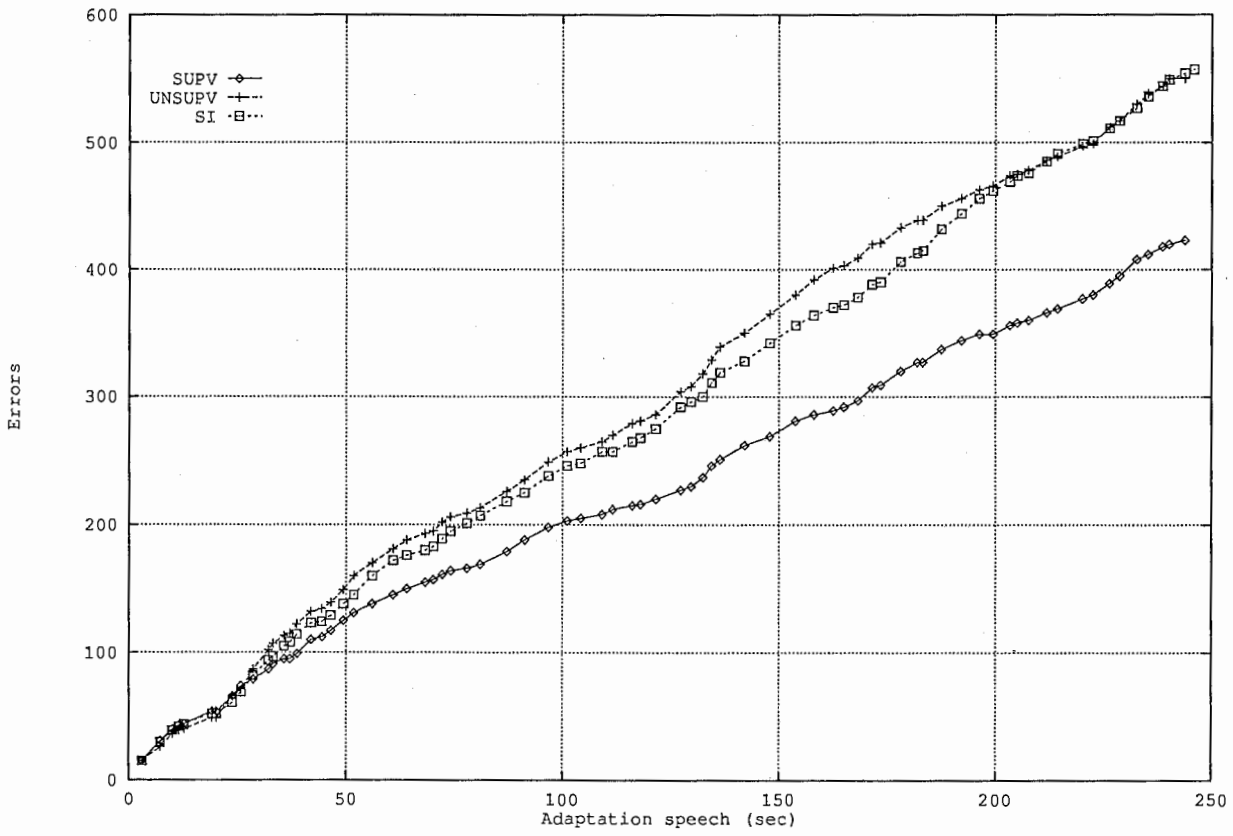
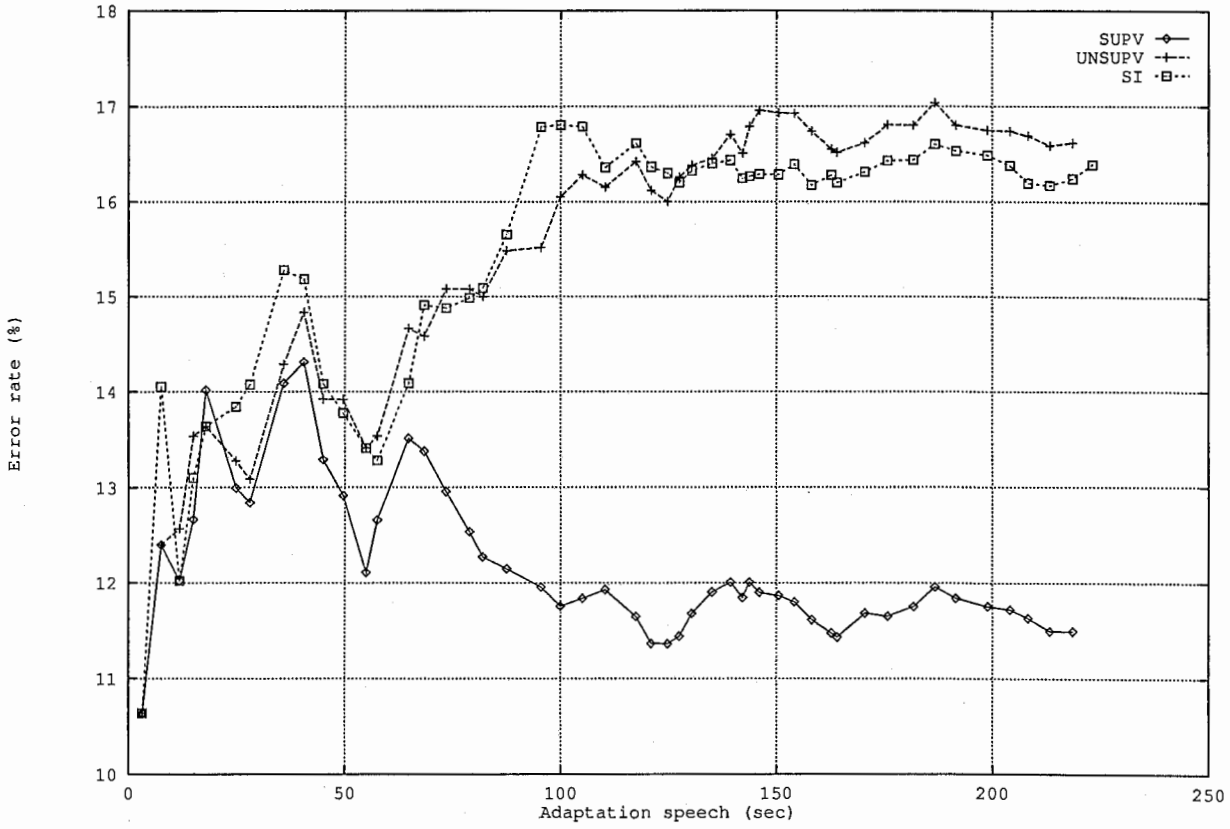


図 6: Incremental Quasi-Bayes Adaptation: Speaker FYOMA



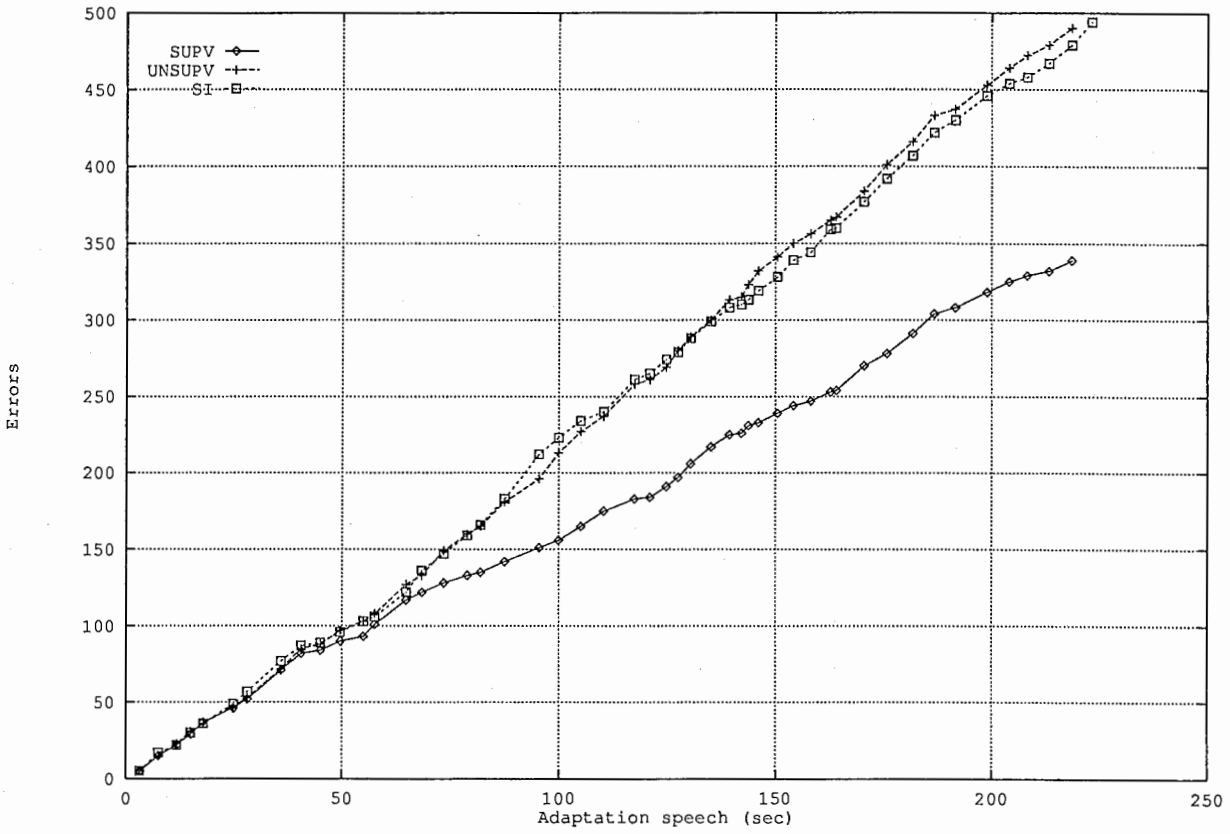
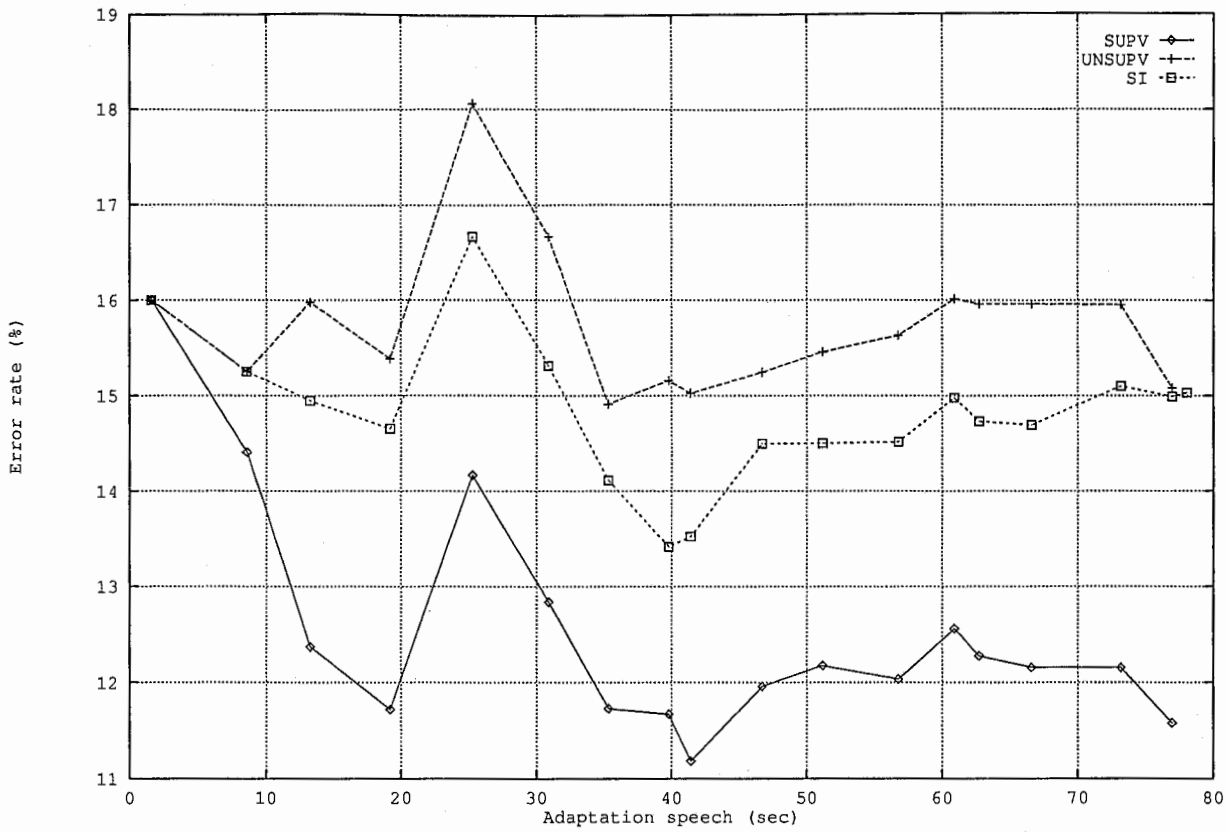


図 7: Incremental Quasi-Bayes Adaptation: Speaker FHITO



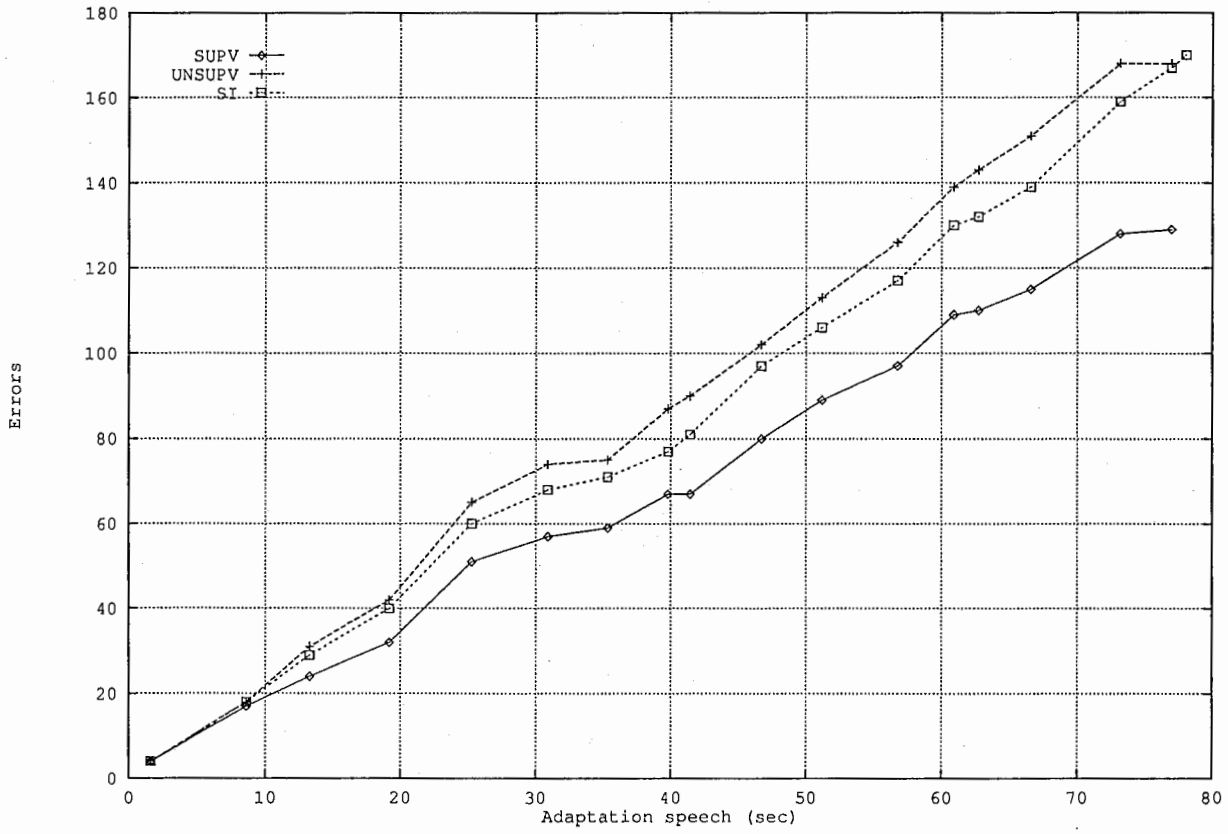
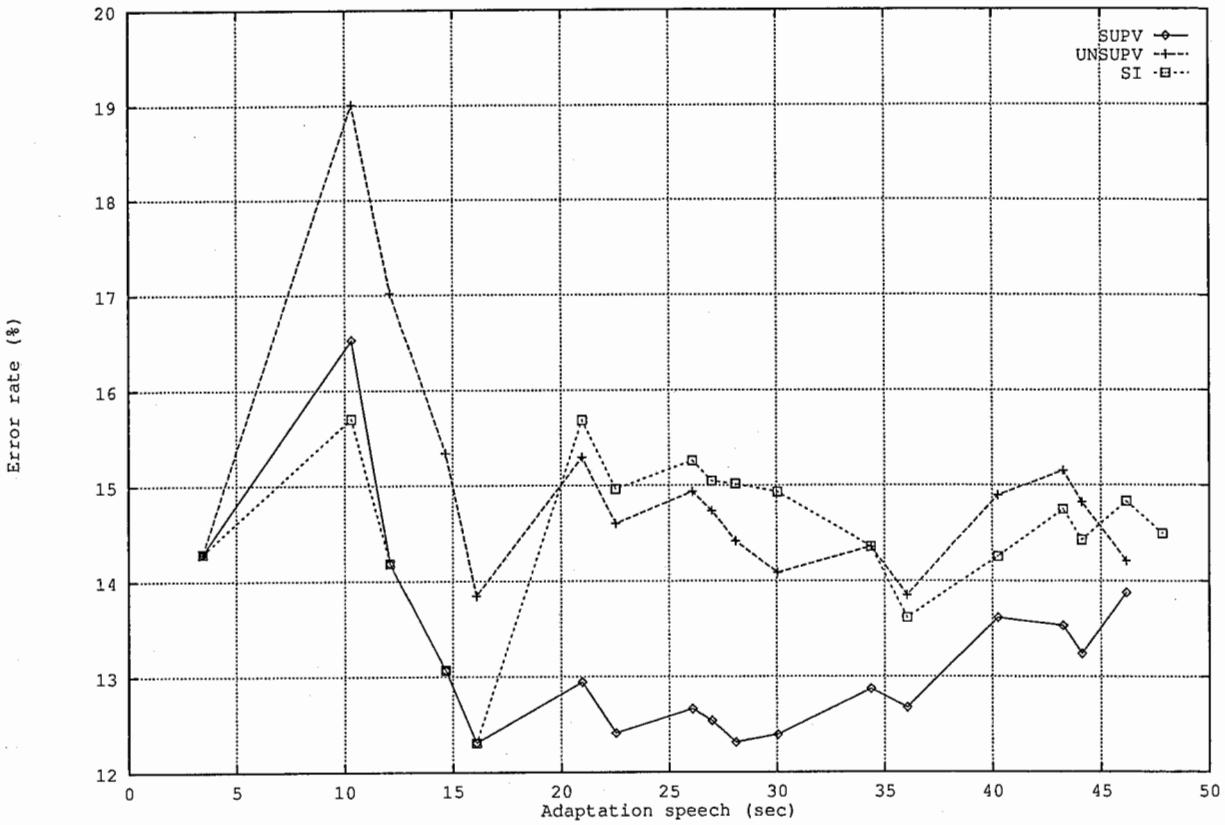


図 8: Incremental Quasi-Bayes Adaptation: Speaker FMAKA



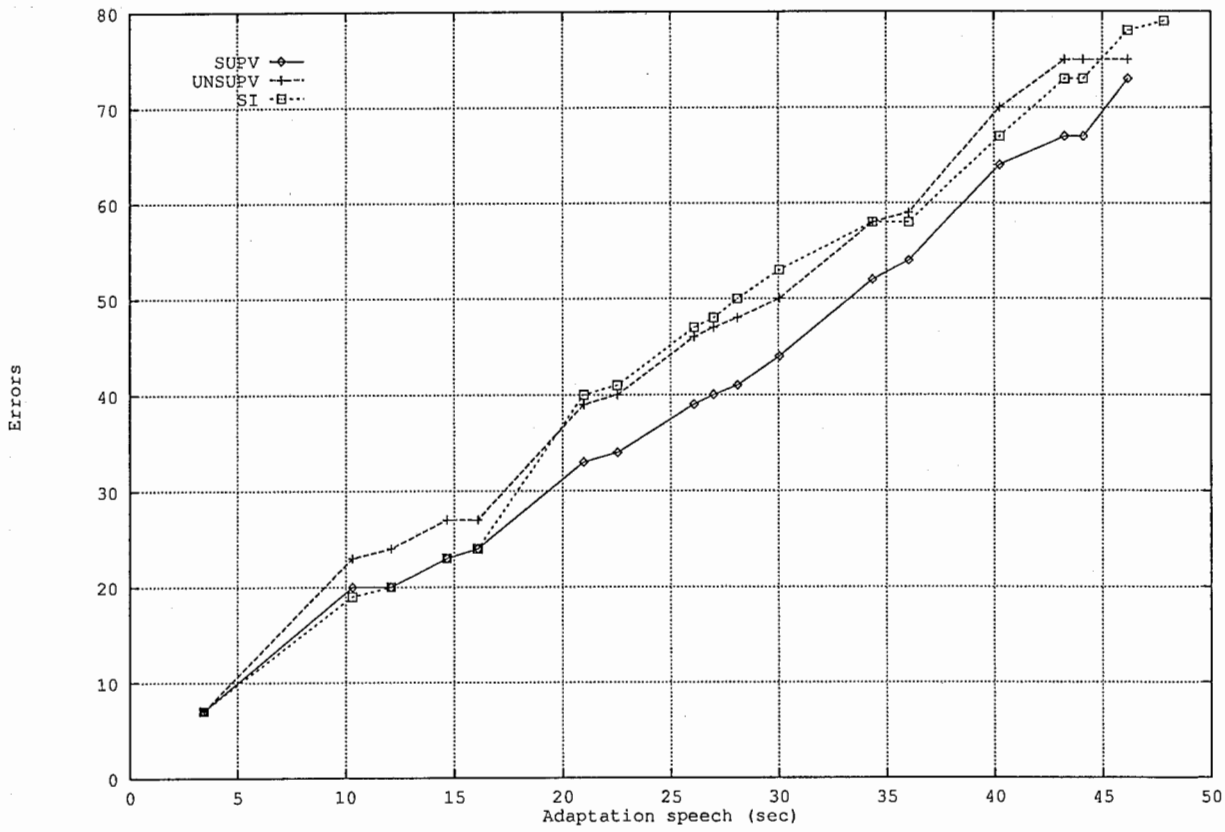
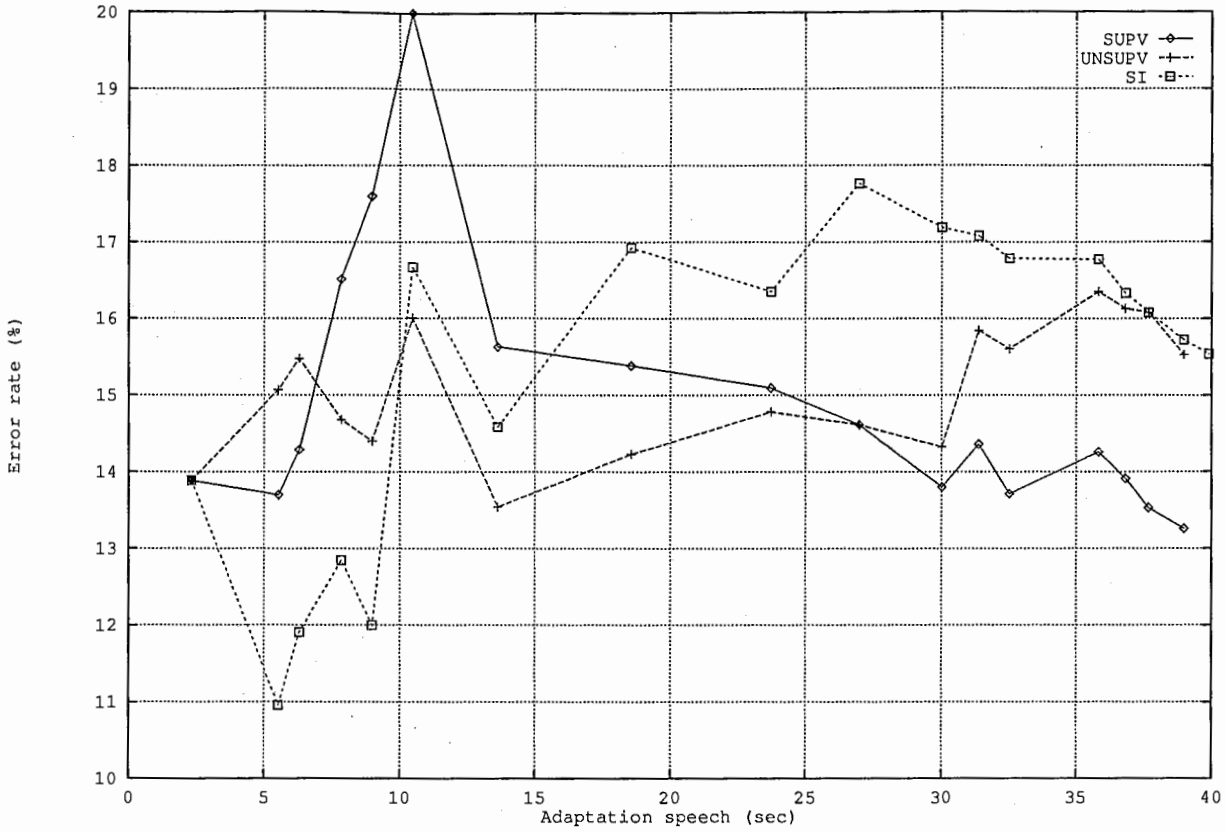


図 9: Incremental Quasi-Bayes Adaptation: Speaker FYOAZ



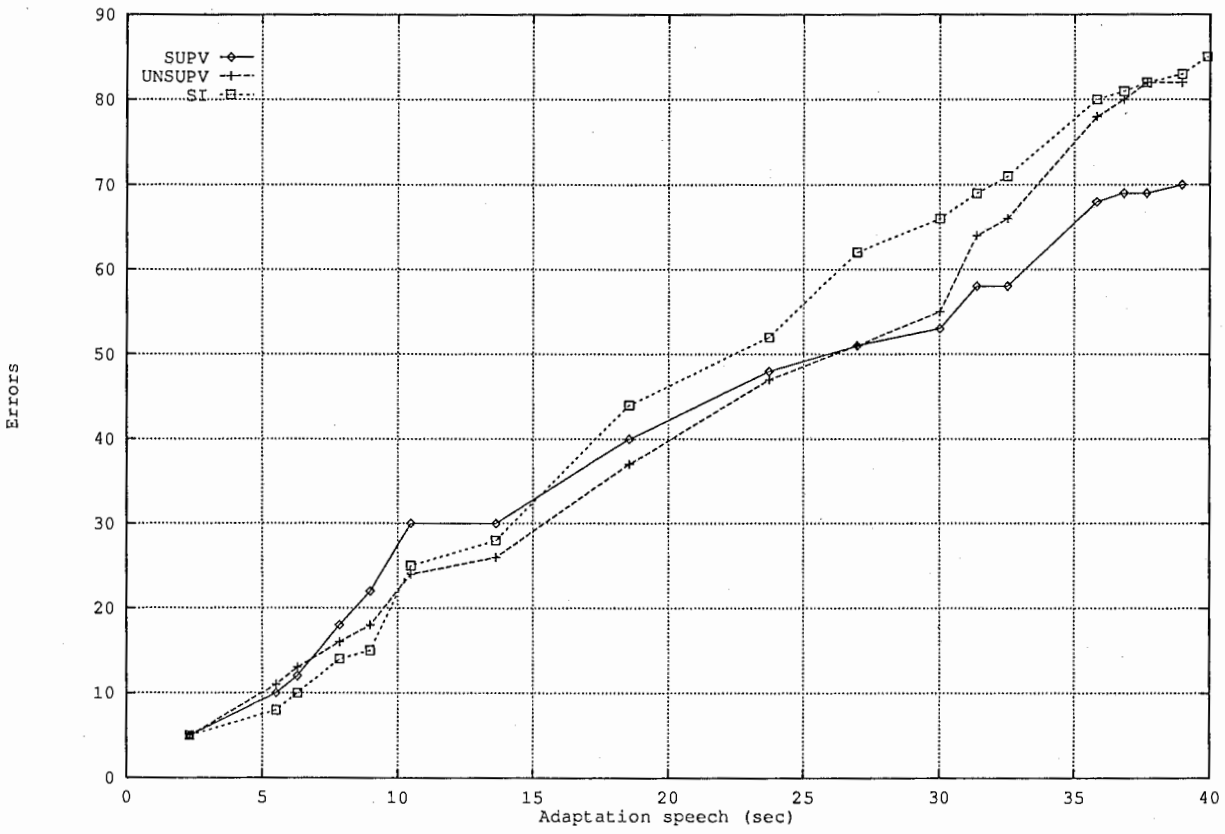
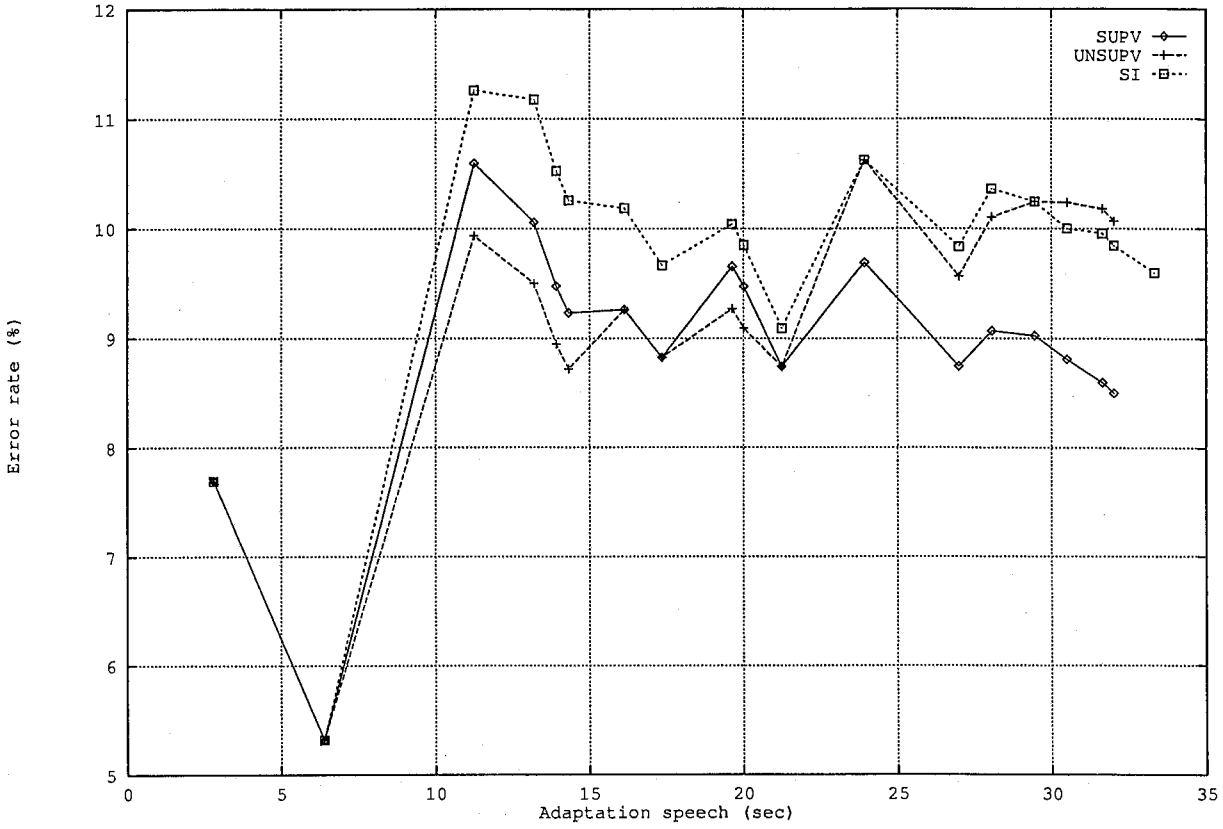


図 10: Incremental Quasi-Bayes Adaptation: Speaker FMASZ



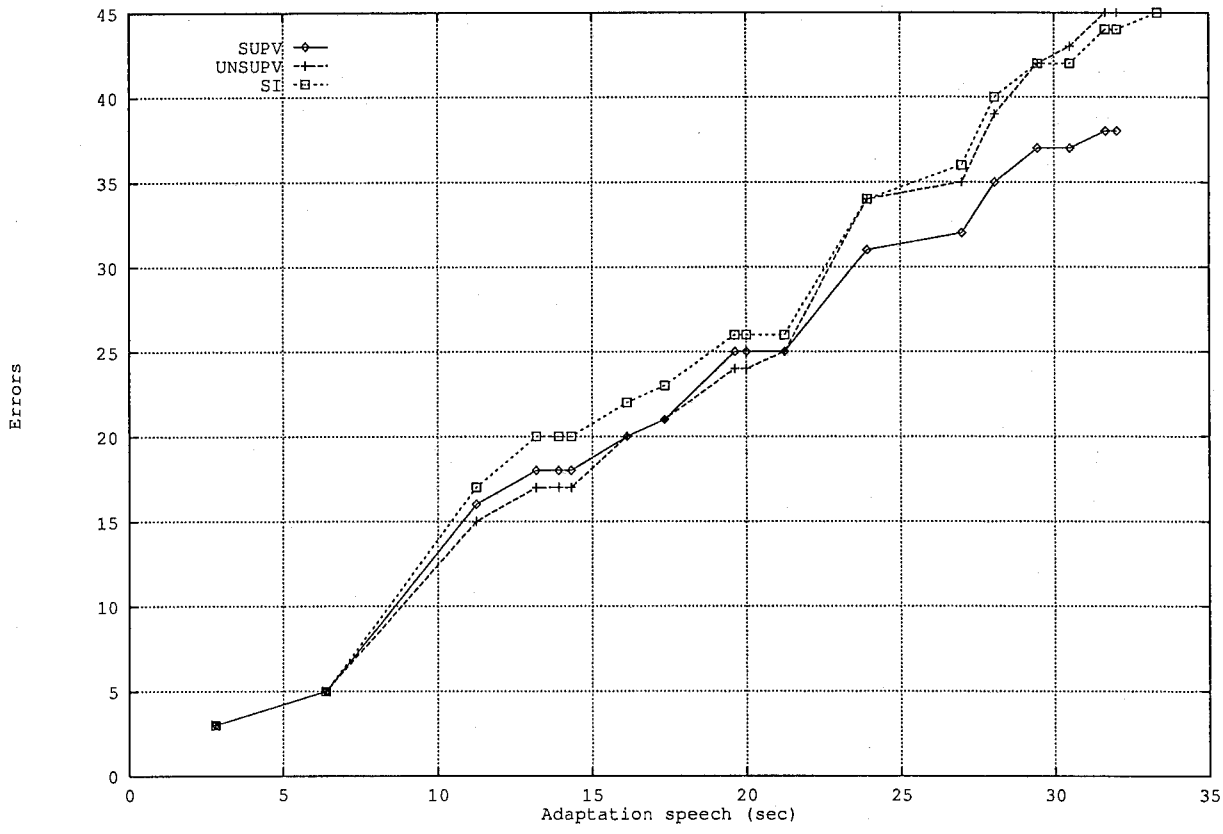
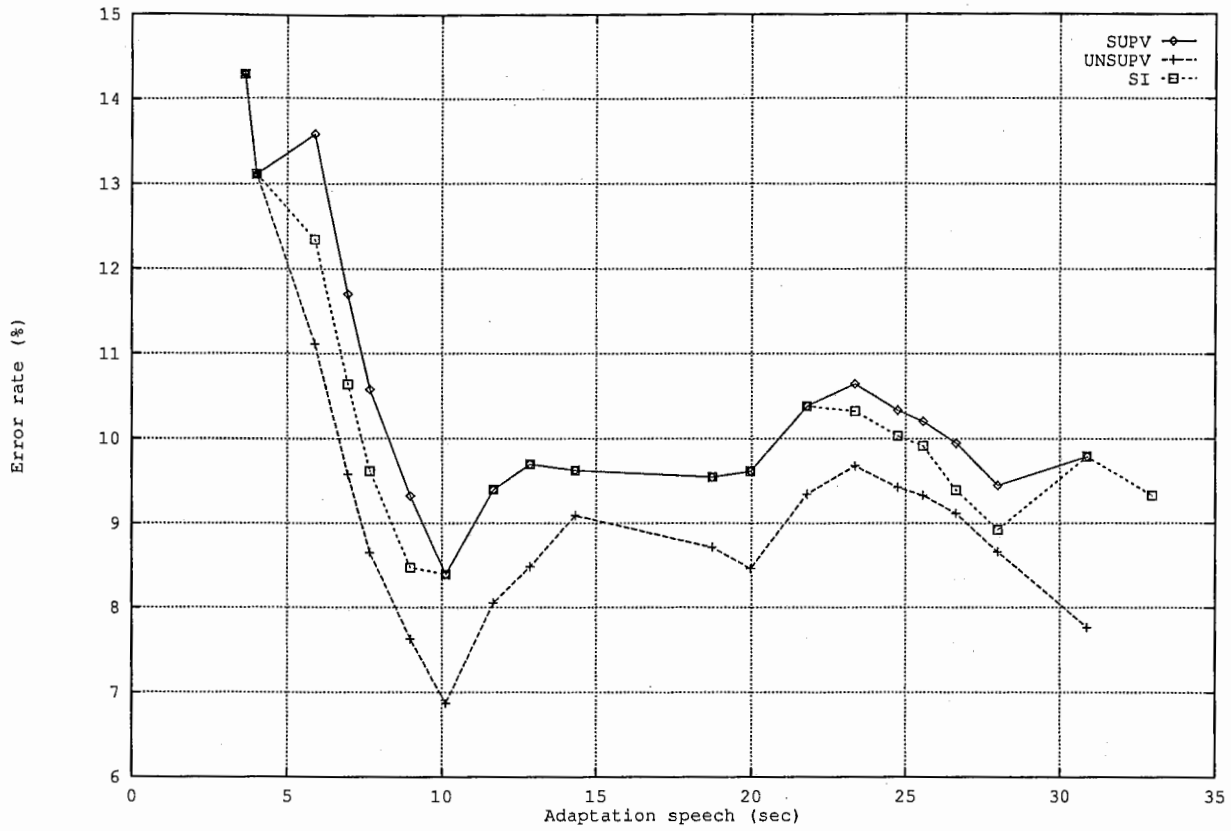


図 11: Incremental Quasi-Bayes Adaptation: Speaker FYUKI



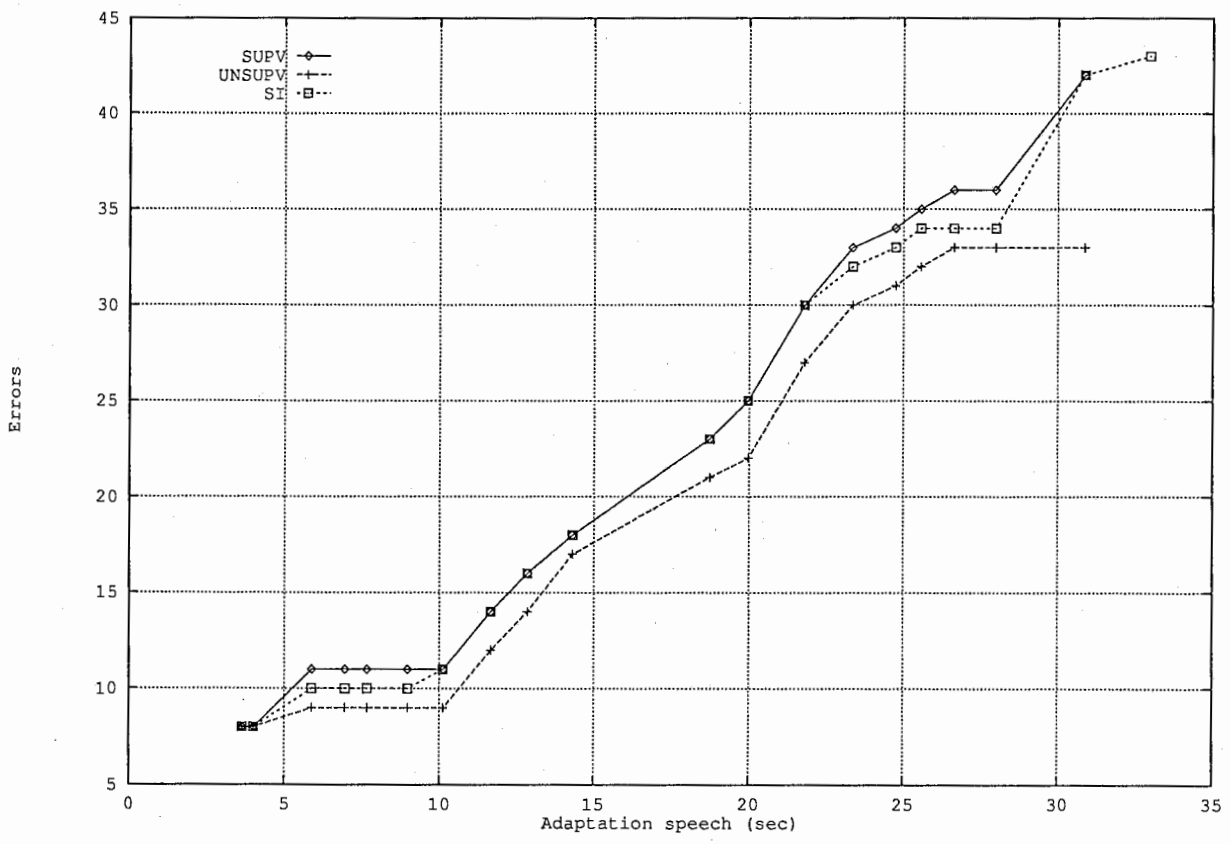
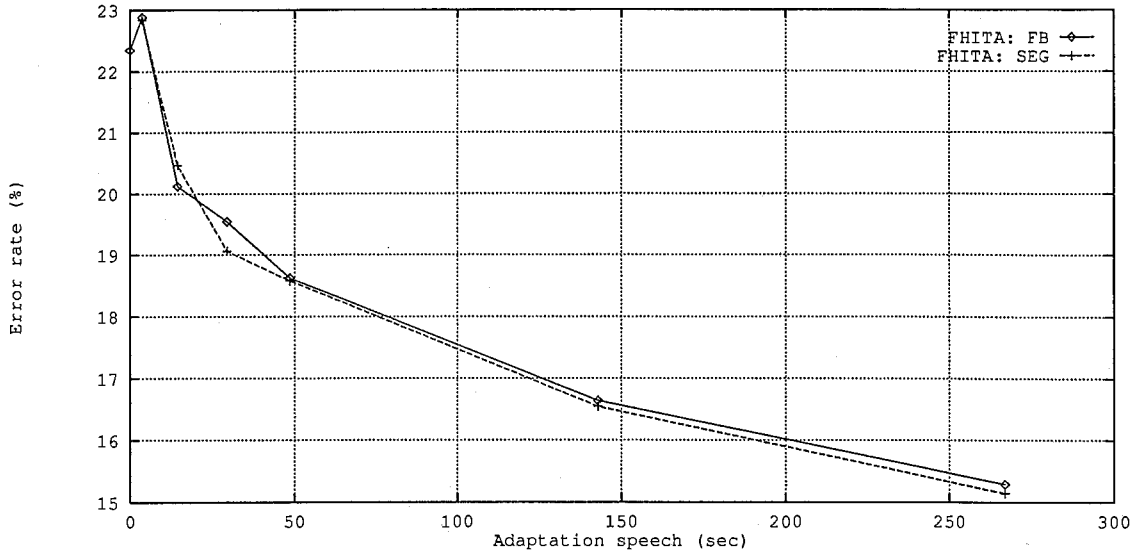
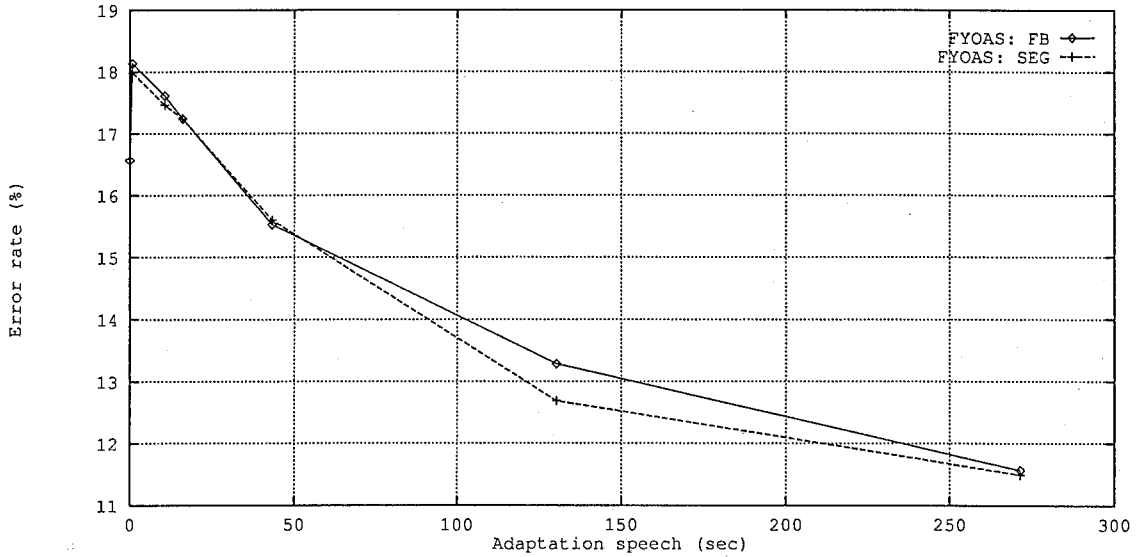
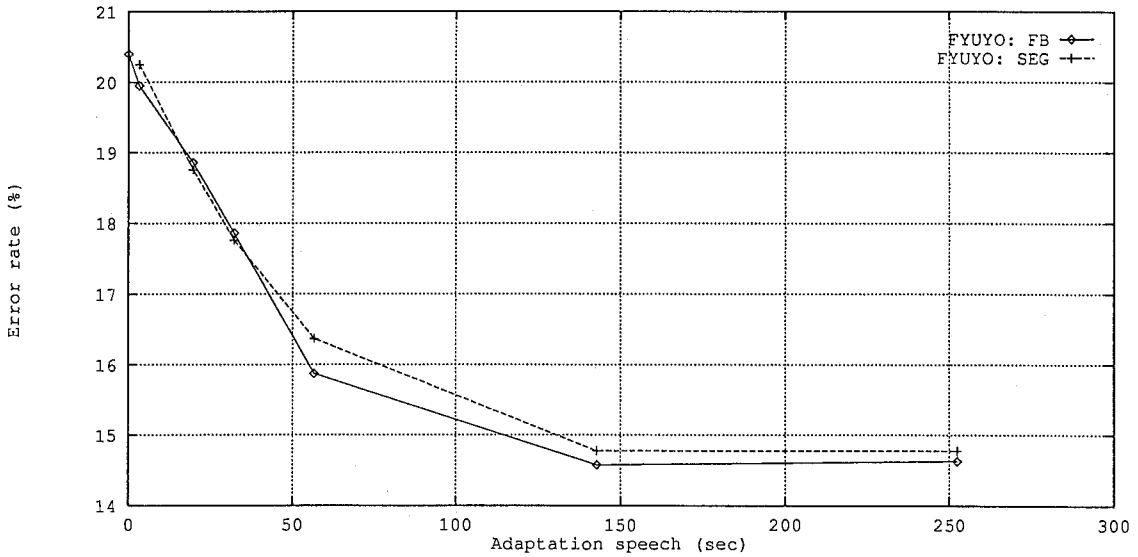


図 12: Incremental Quasi-Bayes Adaptation: Speaker FYOOO







☒ 13: Effect of training method: forward-backward versus segmental (Supervised, Incremental, 1 iteration, Tau = 4)

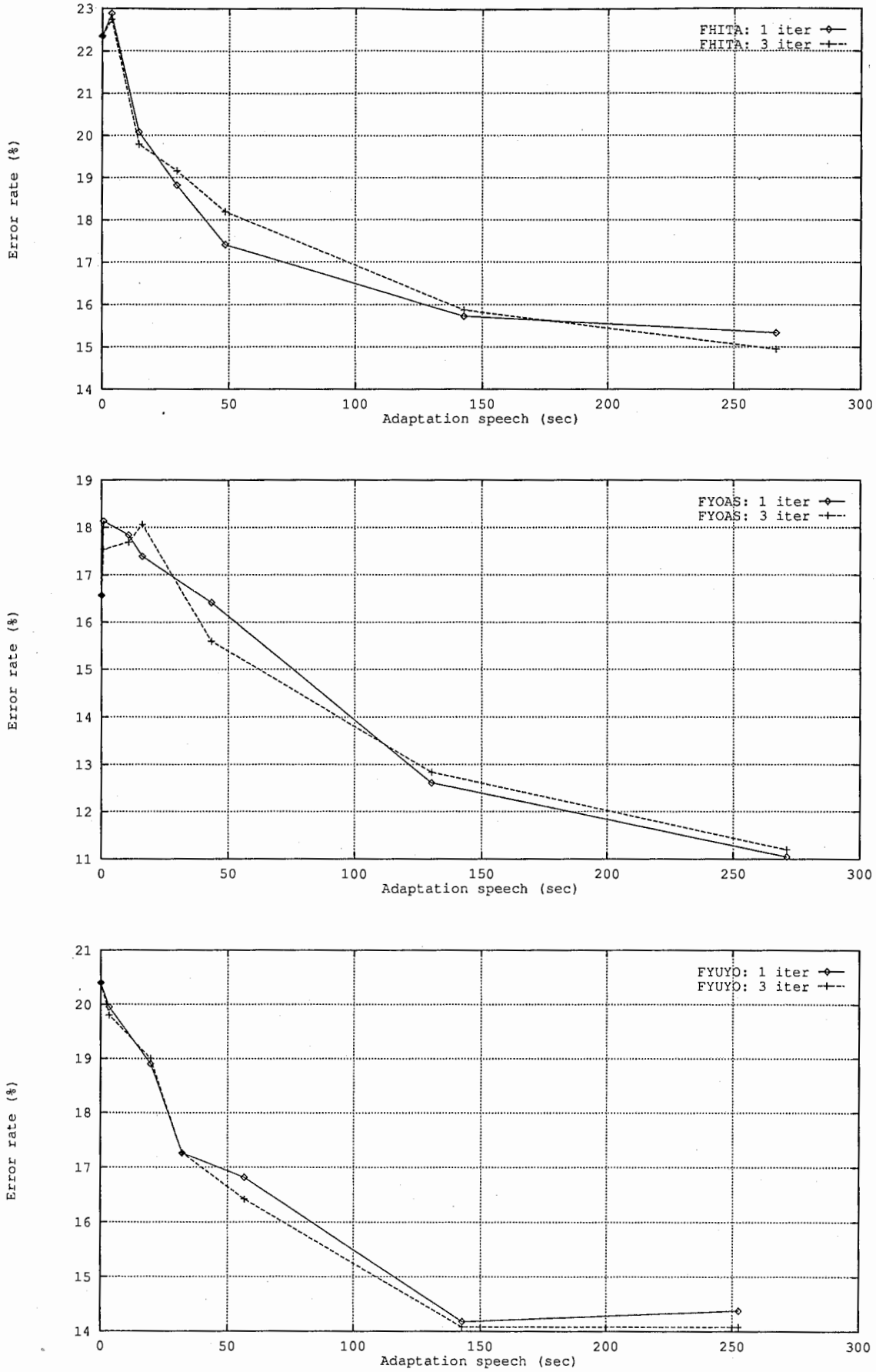


図 14: Effect of additional forward-backward iterations (Supervised, Batch, Tau = 4)

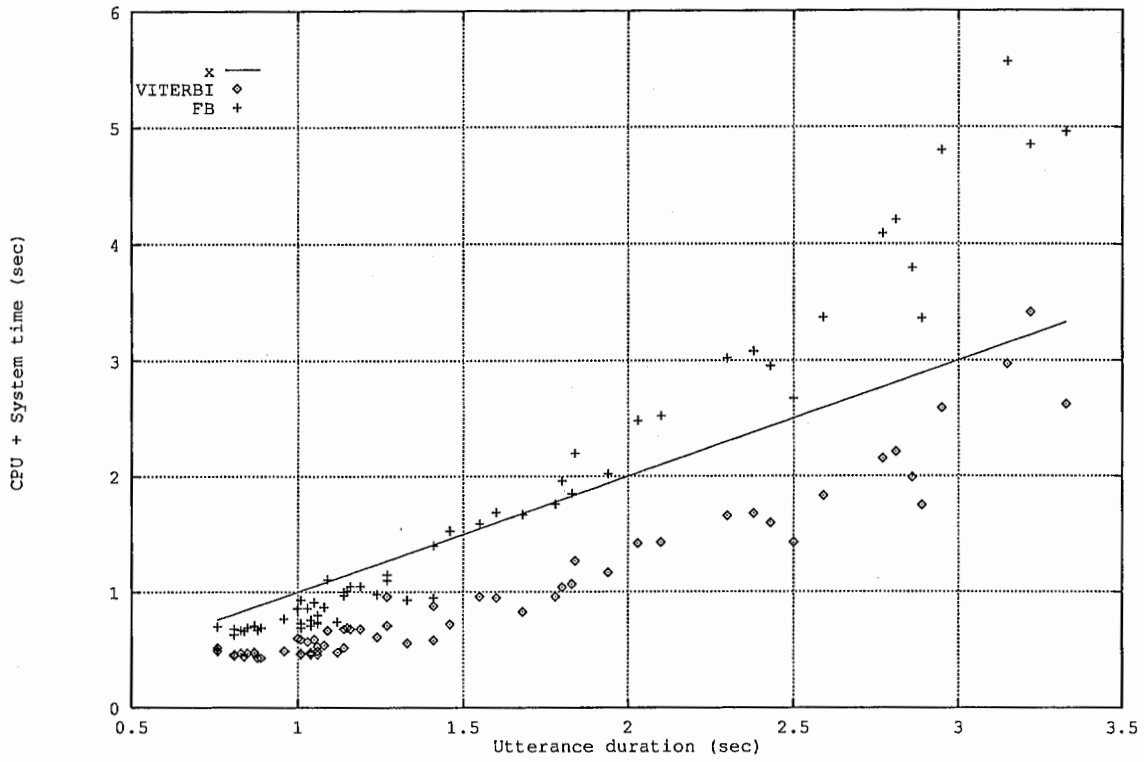


図 15: Computation time: forward-backward versus segmental training (1 iteration, machine: HP 735/125, speaker: FHITA)