

## S1 Stan code

As supplementary material we provide the full *Stan* code of all models. All implementations adhere to the same interface, taking observed returns as input data. Missing data, i.e. which should not be used during fitting, are indicated with a binary mask. Furthermore, all models compute the log likelihood of all returns. On missing returns these correspond to model predictions.

### S1.1 GARCH model

**Listing 1** The code for the GARCH model follows the Stan manual (Stan Development Team 2017). The main change being that we allow for missing return data and include model predictions for these. Furthermore, the log likelihood for every return, observed or missing, is computed in the generated quantities block which is automatically run once after each sampling step.

```

1 data {
2   int<lower=0> T;
3   int<lower=0, upper=1> miss_mask[T];
4   real ret_obs[T]; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7   int N = 0; // number of missing values
8   for (t in 1:T)
9     if (miss_mask[t] == 1) N = N + 1;
10 }
11 parameters {
12   real mu;
13   real<lower=0> alpha0;
14   real<lower=0, upper=1> alpha1;
15   real<lower=0, upper=(1-alpha1)> beta1;
16   real<lower=0> sigma1;
17   real eps_miss[N]; // missing normalized return innovations
18 }
19 transformed parameters {
20   real ret[T]; // returns ... observed or r_t = mu + sigma_t *
      eps_t
21   real<lower=0> sigma[T];
22
23   {
24     int idx = 1; // missing value index
25
26     sigma[1] = sigma1;
27     if (miss_mask[1] == 1) {
28       ret[1] = mu + sigma[1] * eps_miss[idx];
29       idx = idx + 1;
30     } else
31       ret[1] = ret_obs[1];
32
33     for (t in 2:T) {
34       sigma[t] = sqrt(alpha0
35         + alpha1 * pow(ret[t - 1] - mu, 2)
36         + beta1 * pow(sigma[t - 1], 2));

```

```

37     if (miss_mask[t] == 1) {
38         ret[t] = mu + sigma[t] * eps_miss[idx];
39         idx = idx + 1;
40     } else
41         ret[t] = ret_obs[t];
42     }
43 }
44 }
45 model {
46     mu ~ normal(0, 1);
47     sigma1 ~ normal(0, 1);
48
49     ret ~ normal(mu, sigma);
50     // Jacobian correction for transformed innovations
51     for (t in 1:T) {
52         if (miss_mask[t] == 1)
53             target += log(sigma[t]);
54     }
55 }
56 generated quantities {
57     real log_lik[T];
58
59     for (t in 1:T)
60         log_lik[t] = normal_lpdf(ret_obs[t] | mu, sigma[t]);
61 }

```

## S1.2 SV model

**Listing 2** The code for the stochastic volatility model follows the Stan manual (Stan Development Team 2017). As explained there, the latent volatility process is modeled as non-centered.

```

1 data {
2     int<lower=0> T; // time points (equally spaced)
3     int<lower=0, upper=1> miss_mask[T];
4     vector[T] ret_obs; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7     int N = 0; // number of missing values
8     for (t in 1:T)
9         if (miss_mask[t] == 1) N = N + 1;
10 }
11 parameters {
12     real mu_h; // mean log volatility
13     real<lower=-1, upper=1> phi_h; // persistence of volatility
14     real<lower=0> sigma_h; // white noise shock scale
15     vector[T] h_std; // std log volatility at time t
16     vector[N] eps_miss; // missing normalized return
      innovations
17 }
18 transformed parameters {
19     vector[T] h = h_std * sigma_h; // now h ~ normal(0, sigma)
20     real ret[T]; // returns ... observed or r_t =
      sigma_t * eps_t
21     real<lower=0> sigma[T];
22
23     h[1] /= sqrt(1 - phi_h * phi_h); // rescale h[1]

```

```
24 h += mu_h;
25 sigma[1] = exp(h[1] / 2);
26
27 {
28   int idx = 1;
29
30   if (miss_mask[1] == 1) {
31     ret[1] = sigma[1] * eps_miss[idx];
32     idx = idx + 1;
33   } else
34     ret[1] = ret_obs[1];
35
36   for (t in 2:T) {
37     h[t] += phi_h * (h[t-1] - mu_h);
38     sigma[t] = exp(h[t] / 2);
39
40     if (miss_mask[t] == 1) {
41       ret[t] = sigma[t] * eps_miss[idx];
42       idx = idx + 1;
43     } else
44       ret[t] = ret_obs[t];
45   }
46 }
47 }
48 model {
49   phi_h ~ uniform(-1, 1);
50   sigma_h ~ cauchy(0, 5);
51   mu_h ~ cauchy(0, 10);
52   h_std ~ normal(0, 1);
53
54   ret ~ normal(0, sigma);
55   // Jacobian correction for transformed innovations
56   for (t in 1:T) {
57     if (miss_mask[t] == 1)
58       target += h[t] / 2; // = log(sigma[t])
59   }
60 }
61 generated quantities {
62   vector[T] log_lik;
63
64   for (t in 1:T)
65     log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
66 }
```

## S1.3 VS model

**Listing 3** Stan code for the model by Vikram & Sinha. This model is implemented in two specifications: First, as in the original model, with the fundamental price  $p_t^*$  being a running average over past prices.

```

1 data {
2   int<lower=0> T; // time points (equally spaced)
3   int<lower=0, upper=1> miss_mask[T];
4   vector[T] ret_obs; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7   int N = 0; // number of missing values
8   real ret_max = max(ret_obs);
9   real ret_sd = sqrt(variance(ret_obs));
10
11  for (t in 1:T)
12    if (miss_mask[t] == 1) N = N + 1;
13 }
14 parameters {
15   real<lower=0> mu;
16   real<lower=0> lenscale_raw;
17   real<lower=0> sigma_max;
18   real log_p_tau_0;
19   vector[N] eps_miss; // missing normalized return innovations
20 }
21 transformed parameters {
22   real ret[T];
23   real log_p[T]; // log prices
24   real log_p_tau[T];
25   real<lower=0> P_b[T]; // Probability of trading P(|S(t)| = 1)
26   real<lower=0> sigma[T];
27   real<lower=0> lenscale = 1000 * lenscale_raw;
28   real<lower=0, upper=1> tau = exp(- 1 / lenscale);
29
30   log_p[1] = 0; // wlog p_0 = 1
31   // Moving average of prices, i.e. p_tau[i] = tau * p_tau[i-1] +
      (1 - tau) * p[i]
32   log_p_tau[1] = log_mix(tau, log_p_tau_0, log_p[1]);
33   P_b[1] = exp(- mu * fabs(log_p[1] - log_p_tau[1]));
34   sigma[1] = sigma_max * sqrt(2 * P_b[1]);
35   {
36     int idx = 1;
37
38     if (miss_mask[1] == 1) {
39       ret[1] = sigma[1] * eps_miss[idx];
40       idx = idx + 1;
41     } else
42       ret[1] = ret_obs[1];
43
44     for (t in 2:T) {
45       // Note: index shift between prices and returns
46       log_p[t] = log_p[t - 1] + ret[t - 1];
47       log_p_tau[t] = log_mix(tau, log_p_tau[t - 1], log_p[t]);
48       P_b[t] = exp(- mu * fabs(log_p[t] - log_p_tau[t - 1]));
49       sigma[t] = sigma_max * sqrt(2 * P_b[t]);

```

```

50
51     if (miss_mask[t] == 1) {
52         ret[t] = sigma[t] * eps_miss[idx];
53         idx = idx + 1;
54     } else
55         ret[t] = ret_obs[t];
56
57     }
58 }
59 }
60 model {
61     mu ~ gamma(3, 0.03);
62     lenscale_raw ~ inv_gamma(2, 1); // avoid lower boundary ...
63     lenscale ~ inv_gamma(2, 1000)
64     log_p_tau_0 ~ normal(0, 0.1);
65     sigma_max ~ normal(ret_max, ret_max / 4);
66
67     ret ~ normal(0, sigma);
68     // Jacobian correction for transformed innovations
69     for (t in 1:T) {
70         if (miss_mask[t] == 1)
71             target += log(sigma[t]);
72     }
73 }
74 generated quantities {
75     vector[T] log_lik;
76
77     for (t in 1:T)
78         log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
79 }

```

**Listing 4** Secondly, with the fundamental log price  $\log p_t^*$  following a Brownian motion. As explained in the text, in order to increase sampling efficiency this random walk is implemented in non-centered parameters, i.e. innovations.

```

1 data {
2     int<lower=0> T; // time points (equally spaced)
3     int<lower=0, upper=1> miss_mask[T];
4     vector[T] ret_obs; // Note: Masked indices will be treated as
5         missing;
6 }
7 transformed data {
8     int N = 0; // number of missing values
9     real ret_max = max(ret_obs);
10    real ret_sd = sqrt(variance(ret_obs));
11
12    for (t in 1:T)
13        if (miss_mask[t] == 1) N = N + 1;
14 }
15 parameters {
16     real<lower=0> mu;
17     real<lower=0> sigma_max;
18     real log_p_tau_0;
19     vector[N] eps_miss; // missing normalized return innovations
20     // random walk for fundamental price
21     vector[T] log_p_tau_raw;
22     real<lower=0> sigma_p_tau;
23 }

```

```

23 transformed parameters {
24   real ret[T];
25   real log_p[T]; // log prices
26   real log_p_tau[T];
27   real<lower=0> P_b[T]; // Probability of trading P(|S(t)| = 1)
28   real<lower=0> sigma[T];
29
30   log_p[1] = 0; // wlog p_0 = 1
31   // Random walk of fundamental price
32   log_p_tau[1] = log_p_tau_0 + sigma_p_tau * log_p_tau_raw[1];
33   P_b[1] = exp(- mu * fabs(log_p[1] - log_p_tau[1]));
34   sigma[1] = sigma_max * sqrt(2 * P_b[1]);
35   {
36     int idx = 1;
37
38     if (miss_mask[1] == 1) {
39       ret[1] = sigma[1] * eps_miss[idx];
40       idx = idx + 1;
41     } else
42       ret[1] = ret_obs[1];
43
44     for (t in 2:T) {
45       // Note: index shift between prices and returns
46       log_p[t] = log_p[t - 1] + ret[t - 1];
47       log_p_tau[t] = log_p_tau[t - 1] + sigma_p_tau *
log_p_tau_raw[t];
48       P_b[t] = exp(- mu * fabs(log_p[t] - log_p_tau[t - 1]));
49       sigma[t] = sigma_max * sqrt(2 * P_b[t]);
50
51       if (miss_mask[t] == 1) {
52         ret[t] = sigma[t] * eps_miss[idx];
53         idx = idx + 1;
54       } else
55         ret[t] = ret_obs[t];
56
57     }
58   }
59 }
60 model {
61   mu ~ gamma(3, 0.03);
62   log_p_tau_0 ~ normal(0, 0.1);
63   log_p_tau_raw ~ normal(0, 1);
64   sigma_max ~ normal(ret_max, ret_max / 4);
65
66   ret ~ normal(0, sigma);
67   // Jacobian correction for transformed innovations
68   for (t in 1:T) {
69     if (miss_mask[t] == 1)
70       target += log(sigma[t]);
71   }
72 }
73 generated quantities {
74   vector[T] log_lik;
75
76   for (t in 1:T)
77     log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
78 }

```

## S1.4 FW model

**Listing 5** Stan code for the model by Franke & Westerhoff. Agent dynamics follows the DCA-HPM specification and again two specifications are assumed for the fundamental price dynamics. Here, the fundamental price is computed as a running average over past price. Note that as in the original model, the *log* fundamental price is denoted by  $p_t^*$ .

```

1 data {
2   int<lower=0> T; // time points (equally spaced)
3   int<lower=0, upper=1> miss_mask[T];
4   vector[T] ret_obs; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7   int N = 0; // number of missing values
8   real ret_sd = sqrt(variance(ret_obs));
9   // mu and beta fixed ... redundant anyways
10  real mu = 0.01;
11  real beta = 1.0;
12
13  for (t in 1:T)
14    if (miss_mask[t] == 1) N = N + 1;
15 }
16 parameters {
17   real<lower=0> phi;
18   real<lower=0> xi;
19   real alpha_0;
20   real<lower=0> alpha_n;
21   real<lower=0> alpha_p;
22   real<lower=0> sigma_f;
23   real<lower=0> sigma_c;
24   real<lower=0, upper=1> n_f_1;
25   real<lower=0> lenscale_raw;
26   real p_star_0;
27   vector[N] eps_miss; // missing normalized return innovations
28 }
29 transformed parameters {
30   vector[T] n_f;
31   vector[T] demand;
32   vector[T] sigma;
33   // Note: All prices are actually log prices!
34   vector[T] p_star;
35   vector[T] p;
36   real ret[T];
37   real<lower=0> lenscale = 1000 * lenscale_raw;
38   real<lower=0, upper=1> tau = exp(- 1 / lenscale);
39
40   p[1] = 0; // wlog log p_1 = 0
41   p_star[1] = log_mix(tau, p_star_0, p[1]);
42
43   n_f[1] = n_f_1;
44   demand[1] = 0;
45   sigma[1] = mu * sqrt( square(n_f[1] * sigma_f)
46     + square((1 - n_f[1]) * sigma_c));
47   {
48     int idx = 1;
49

```

```

50   if (miss_mask[1] == 1) {
51     ret[1] = sigma[1] * eps_miss[idx];
52     idx = idx + 1;
53   } else
54     ret[1] = ret_obs[1];
55
56   for (t in 2:T) {
57     // Note: index shift between prices and returns
58     p[t] = p[t - 1] + ret[t - 1];
59     p_star[t] = log_mix(tau, p_star[t-1], p[t]);
60
61     {
62     // equation (HPM)
63     real a = alpha_n * (n_f[t-1] - (1 - n_f[t-1]))
64       + alpha_0
65       + alpha_p * square(p[t-1] - p_star[t-1]);
66     // equation (DCA)
67     n_f[t] = inv_logit(beta * a);
68     demand[t] = mu * (n_f[t] * phi * (p_star[t] - p[t])
69       + (1 - n_f[t]) * xi * (p[t] - p[t-1]));
70     // structured stochastic volatility
71     sigma[t] = mu * sqrt( square(n_f[t] * sigma_f
72       + square((1 - n_f[t]) * sigma_c));
73     }
74
75     if (miss_mask[t] == 1) {
76       ret[t] = sigma[t] * eps_miss[idx];
77       idx = idx + 1;
78     } else
79       ret[t] = ret_obs[t];
80   }
81 }
82 }
83 model {
84   phi ~ student_t(5, 0, 1);
85   xi ~ student_t(5, 0, 1);
86   alpha_0 ~ student_t(5, 0, 1);
87   alpha_n ~ student_t(5, 0, 1);
88   alpha_p ~ student_t(5, 0, 1);
89   sigma_f ~ normal(0, ret_sd / mu);
90   sigma_c ~ normal(0, 2.0 * ret_sd / mu);
91   p_star_0 ~ normal(0, 0.2);
92   lenscale_raw ~ inv_gamma(2, 1); // avoid lower boundary ...
93   lenscale ~ inv_gamma(2, 1000)
94
95   // Price likelihood
96   ret ~ normal(demand, sigma);
97   // Jacobian correction for transformed innovations
98   for (t in 1:T) {
99     if (miss_mask[t] == 1)
100       target += log(sigma[t]);
101   }
102 }
103 generated quantities {
104   vector[T] log_lik;
105
106   for (t in 1:T)
107     log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);

```



107 }

**Listing 6** Stan code for the model by Franke & Westerhoff with the log fundamental price  $p_t^*$  following a random walk.

```

1 data {
2   int<lower=0> T; // time points (equally spaced)
3   int<lower=0, upper=1> miss_mask[T];
4   vector[T] ret_obs; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7   int N = 0; // number of missing values
8   real ret_sd = sqrt(variance(ret_obs));
9   // mu and beta fixed ... redundant anyways
10  real mu = 0.01;
11  real beta = 1.0;
12
13  for (t in 1:T)
14    if (miss_mask[t] == 1) N = N + 1;
15 }
16 parameters {
17   real<lower=0> phi;
18   real<lower=0> xi;
19   real alpha_0;
20   real<lower=0> alpha_n;
21   real<lower=0> alpha_p;
22   real<lower=0> sigma_f;
23   real<lower=0> sigma_c;
24   real<lower=0, upper=1> n_f_1;
25   // p_star random walk in non-centered parameterization
26   vector[T] epsilon_star;
27   real<lower=0> sigma_p_star;
28   vector[N] eps_miss; // missing normalized return innovations
29 }
30 transformed parameters {
31   vector[T] n_f;
32   vector[T] demand;
33   vector[T] sigma;
34   // Note: All prices are actually log prices!
35   vector[T] p_star;
36   vector[T] p;
37   real ret[T];
38
39   p[1] = 0; // wlog log p_1 = 0
40   p_star[1] = p[1] + epsilon_star[1]; // fixme ... interpretation
      epsilon_raw[1]
41
42   n_f[1] = n_f_1;
43   demand[1] = 0;
44   sigma[1] = mu * sqrt( square(n_f[1] * sigma_f)
45     + square((1 - n_f[1]) * sigma_c));
46   {
47     int idx = 1;
48
49     if (miss_mask[1] == 1) {
50       ret[1] = sigma[1] * eps_miss[idx];
51       idx = idx + 1;

```

```

52 } else
53   ret[1] = ret_obs[1];
54
55   for (t in 2:T) {
56     // Note: index shift between prices and returns
57     p[t] = p[t - 1] + ret[t - 1];
58     p_star[t] = p_star[t-1] + sigma_p_star * epsilon_star[t];
59
60     {
61     // equation (HPM)
62     real a = alpha_n * (n_f[t-1] - (1 - n_f[t-1]))
63       + alpha_0
64       + alpha_p * square(p[t-1] - p_star[t-1]);
65     // equation (DCA)
66     n_f[t] = inv_logit(beta * a);
67     demand[t] = mu * (n_f[t] * phi * (p_star[t] - p[t])
68       + (1 - n_f[t]) * xi * (p[t] - p[t-1]));
69     // structured stochastic volatility
70     sigma[t] = mu * sqrt( square(n_f[t] * sigma_f)
71       + square((1 - n_f[t]) * sigma_c));
72     }
73
74     if (miss_mask[t] == 1) {
75       ret[t] = sigma[t] * eps_miss[idx];
76       idx = idx + 1;
77     } else
78       ret[t] = ret_obs[t];
79   }
80 }
81 }
82 model {
83   phi ~ student_t(5, 0, 1);
84   xi ~ student_t(5, 0, 1);
85   alpha_0 ~ student_t(5, 0, 1);
86   alpha_n ~ student_t(5, 0, 1);
87   alpha_p ~ student_t(5, 0, 1);
88   sigma_f ~ normal(0, ret_sd / mu);
89   sigma_c ~ normal(0, 2.0 * ret_sd / mu);
90   epsilon_star ~ normal(0, 1);
91   sigma_p_star ~ normal(0, ret_sd / 2.0);
92
93   // Price likelihood
94   ret ~ normal(demand, sigma);
95   // Jacobian correction for transformed innovations
96   for (t in 1:T) {
97     if (miss_mask[t] == 1)
98       target += log(sigma[t]);
99   }
100 }
101 generated quantities {
102   vector[T] log_lik;
103
104   for (t in 1:T)
105     log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
106 }

```

## S1.5 ALW model

**Listing 7** Stan code for the model by Alfarano, Lux & Wagner.

```

1 data {
2   int<lower=0> T; // time points (equally spaced)
3   int<lower=0, upper=1> miss_mask[T];
4   vector[T] ret_obs; // Note: Masked indices will be treated as
      missing;
5 }
6 transformed data {
7   int N = 0; // number of missing values
8   real T_quot = 1; // wlog fixed at 1
9
10  for (t in 1:T)
11    if (miss_mask[t] == 1) N = N + 1;
12 }
13 parameters {
14   vector<lower=-1, upper=1>[T + 1] x; // sentiment over time
15   real<lower=0> sigma_f;
16   real<lower=0> alpha;
17   real<lower=0> beta;
18   vector[N] eps_miss; // missing normalized return innovations
19 }
20 transformed parameters {
21   vector[T] ret;
22
23   {
24     int idx = 1;
25
26     for (t in 1:T) {
27       if (miss_mask[t] == 1) {
28         ret[t] = T_quot * (x[t + 1] - x[t]) + sigma_f * eps_miss[
29           idx];
30         idx = idx + 1;
31       } else
32         ret[t] = ret_obs[t];
33     }
34   }
35 model {
36   sigma_f ~ normal(0, 1);
37   alpha ~ normal(0, 1);
38   beta ~ normal(0, 1);
39
40   // x[1] implicitly uniform
41   for (t in 2:(T+1))
42     x[t] ~ normal(x[t - 1] - 2.0 * alpha * x[t - 1],
43                 sqrt(2.0 * beta * (1 - square(x[t - 1]))));
44
45   // Price likelihood
46   ret ~ normal(T_quot * (x[2:(T+1)] - x[1:T]), sigma_f);
47   // Jacobian correction for transformed innovations
48   for (t in 1:T) {
49     if (miss_mask[t] == 1)
50       target += log(sigma_f);
51   }

```

```
52 }
53 generated quantities {
54     vector[T] sigma;
55     vector[T] log_lik;
56
57     for (t in 1:T) {
58         sigma[t] = sqrt(square(T_quot * (x[t + 1] - x[t])) + square(
59             sigma_f));
60         log_lik[t] = normal_lpdf(ret_obs[t] | T_quot * (x[t + 1] - x[t
61             ]), sigma_f);
62     }
63 }
```