SAEM 算法在处理逻辑回归缺失值中的应用

黄脉，夏亦凡，周靖智

1 引言

EM 算法在处理缺失值时具有非常广泛的应用，而逻辑回归作为一种有监督学习中的常见分类方法，在存在缺失值情况下回归效率并不理想。本文在 EM 算法的基础上，提出了一种基于 Metroplics-Hasting 抽样的 EM 算法的随机近似版本 (SAEM)，用于对具有缺失数据的逻辑回归进行统计推断。在数值模拟阶段，我们利用 Monte-Carlo 方法研究 SAEM 算法的收敛性，比较不同算法的表现差异，在不同确实机制下的表现以及模型选择等问题，发现 SAEM 算法计算效率较高，并具有良好的覆盖范围和变量选择特性。

2 研究背景

缺失数据存在于几乎所有的数据研究领域中，一种处理缺失值的常用方法就是处理估计过程，使之能够应用于不完整数据，例如 EM 算法通过补全潜变量来获得最大似然估计，但其中 E-step 关于缺失值的期望在部分情况下难以求得，故 EM 算法适用于特定的推断问题，但对于逻辑回归这种简单模型的缺失值问题，处理便较为困难。在广义线性模型中，有学者提出使用蒙特卡洛 EM（MCEM）算法, 即使用蒙特卡洛抽样的经验总和来代替期望积分，并且使用蒙特卡洛版本的 Louis 公式来估计方差。但他们的方法计算成本高昂，效率较低，并且只考虑了在数据集中两个变量存在缺失值的情况。而本文提出一种随机近似 EM 算法 (SAEM) 作为 MCEM 的替代方法。SAEM 采用随机近似来估计完全数据似然的条件期望，而不是生成大量的蒙特卡洛样本，故而 SAEM 在计算上具有较大的优势，同时，它还允许使用基于惩罚观测似然的标准进行模型选择，在实践中有较大效用。

3 SAEM 算法

3.1 假设与符号

设 (y, x) 为观测数据，其中 $y = (y_i, 1 \leq i \leq n)$ 是一个 n 维的二元响应向量，$y_i \in 0, 1\ x = x_i j, 1 \leq i \leq n, 1 \leq j \leq n)$ 是一个 $n \times p$ 的实矩阵，二分类的逻辑回归模型可

以写为:

$$\mathbb{P}(y_i = 1|x_i;\beta) = \frac{\exp(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij})}{1+\exp(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij})}, \quad i = 1,\ldots,n \tag{1}$$

其中 $x_{i1},...,x_{ip}$ 是关于 i 的变量,$\beta_0,\beta_1,...,\beta_p$ 是未知参数。我们假设 $x_i = (x_{i1},...,x_{ip})$ 服从正态分布:

$$x_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}_p(\mu,\Sigma), \quad i = 1,\ldots,n \tag{2}$$

设 $= (\mu,\Sigma,\beta)$ 为模型的参数集, 那么完整数据的似然函数为:

$$\begin{aligned}\mathcal{LL}(\theta;x,y) &= \sum_{i=1}^{n}\mathcal{LL}(\theta;x_i,y_i)\\ &= \sum_{i=1}^{n}\left(\log(\text{ı}(y_i|x_i;\beta)) + \log(\text{ı}(x_i;\mu,\Sigma))\right)\end{aligned} \tag{3}$$

我们的主要目标是在设计阵 x 存在缺失值的情况下，估计参数向量 $\beta = (\beta_i, 1 \le i \le n)$。对于每个 i，我们用 $x_{i,obs}$ 表示 xi 中观测到的元素，用 $x_{i,mis}$ 表示缺失的元素, 则我们可将设计阵分解为 $x = (x_{obs}\ x_{mis})$，注意不同个体缺失的元素可能不同。
对于每个 i，我们定义缺失数据的示性向量 $r_i = (r_{ij}, 1 \le j \le n)$，如果 x_{ij} 缺失，则 $r_{ij} = 1$，否则 $r_{ij} = 0$。矩阵 $r = (r_i, 1 \le i \le n))$ 则定义了缺失数据模式。缺失数据的机制由给定 x 和 y 条件下 r 的条件分布来描述，参数为 Φ，即 $p(r_i|x_i\ y_i\ \Phi)$。在本文中，我们假设缺失数据随机（MAR）机制，这意味着缺失值机制可以被忽略,θ 的最大似然估计可以通过最大化 $\mathcal{LL}(\theta;y,x_{obs})$ 来获得。

3.2 EM 与 MCEM

在 EM 与 MCEM 算法中，我们的目标是通过最大化对数似然 $\mathcal{LL}(\theta;y,x_{obs})$ 来估计逻辑回归模型的参数。我们从经典的 EM 公式开始，从不完整数据中获得最大似然估计。给定某个初始值 θ_0, 通过以下两个步骤将 θ_{k-1} 迭代更新为 θ_k:

E-Step：计算

$$\begin{aligned}Q_k(\theta) &= \mathbb{E}[\mathcal{LL}(\theta;x,y)|x_{\text{obs}},y;\theta_{k-1}]\\ &= \int \mathcal{LL}(\theta;x,y)\text{ı}(x_{\text{miss}}|x_{\text{obs}},y;\theta_{k-1})dx_{\text{miss}}\end{aligned} \tag{4}$$

M-Step：更新 θ 的 MLE 估计:

$$\theta_k = \arg\max_{\theta} Q_k(\theta) \tag{5}$$

由于逻辑回归模型的 E-Step 中的期望 (4) 没有显式表达式，可以使用 MCEM 估计 (4) 式:E-Step 利用蒙特卡洛方法从目标分布

$$p\left(x_{i,\text{ miss}} \mid x_{i,\text{ obs}}, y_i; \theta_{k-1}\right) \tag{6}$$

中生成多个缺失数据样本，并用经验均值近似估计完全对数似然的期望。但是，对 E-Step 进行蒙特卡洛近似可能需要大量的计算工作，效率较低。为了提高计算效率，我们用基于单次模拟生成的 x_{mis} 随机近似估计 E-Step(4) 式。初始化 θ_0 后，第 k 次迭代包括以下三个步骤：

模拟：对于 $i = 1, 2, ..., n$, 从

$$p\left(x_{i,\text{ miss}} \mid x_{i,\text{ obs}}, y_i; \theta_{k-1}\right) \tag{7}$$

中抽取 $x_{i,mis}^{(k)}$。

随机近似：根据

$$Q_k(\theta) = Q_{k-1}(\theta) + \gamma_k \left(\mathcal{LL}(\theta; x_{\text{obs}}, x_{\text{miss}}^{(k)}, y) - Q_{k-1}(\theta) \right) \tag{8}$$

更新函数 Q，其中 γ_k 是递减的正数序列.

最大化：更新 θ 的 MLE 估计：

$$\theta_k = \arg\max_{\theta} Q_k(\theta) \tag{9}$$

在 (8) 中序列 γ_K 的选择对于保证 SAEM 收敛性非常重要。我们将在之后的数值模拟中看到，使用 $\gamma_k = 1$ 进行前几次迭代，之后使用随 $1/k$ 递减的序列，可以获得非常好的收敛效果。

3.3 Metropolis-Hasting

在逻辑回归的情况下，传统 EM 与 MCEM 算法的缺陷是我们无法从其条件分布 $p(x_{i,mis}|x_{i,obs}, y; \theta_{k-1})(7)$ 中精确抽取到未观测数据，这是因为该式并没有显示表示。而我们可以使用 Metropolis-Hastings（MH）算法，通过构建一个平稳分布与目标分布相同的马尔可夫链。经过 M 次迭代后该链的状态则可作为来自目标分布的样本。为了定义 MH 算法的建议分布，我们看到目标分布 $p(x_{i,mis}|x_{i,obs}, y; \theta_{k-1})(7)$ 可以分解如下：

$$p(x_{i,\text{miss}}|x_{i,\text{obs}}, y_i; \theta) \propto p(y_i|x_i; \beta)p(x_{i,\text{miss}}|x_{i,\text{obs}}; \mu, \Sigma) \tag{10}$$

我们选择建议分布为 $p(x_{i,mis}|x_{i,obs}, \mu, \Sigma)$，有正态分布的的条件分布公式可知

$$x_{i,\text{miss}}|x_{i,\text{obs}} \sim \mathcal{N}_p(\mu_i, \Sigma_i) \tag{11}$$

其中

$$\mu_i = \mu_{i,\text{miss}} + \Sigma_{i,\text{miss,obs}}\Sigma_{i,\text{obs,obs}}^{-1}(x_{i,\text{obs}} - \mu_{i,\text{obs}}) \tag{12}$$

$$\Sigma_i = \Sigma_{i,\text{miss,miss}} - \Sigma_{i,\text{miss,obs}}\Sigma_{i,\text{obs,obs}}^{-1}\Sigma_{i,\text{obs,miss}} \tag{13}$$

$_{i,\text{mis}}$ 是 u_i 的缺失元素。协方差矩阵 Σ 也以相同方式求得。具体算法流程如下:

Algorithm 1: Metropolis-Hastings sampling.

Input: An initial sample $x_i^{(0)} \sim g(x_{i,\text{miss}})$;
for $s = 1, 2, \ldots, S$ **do**
  Generate $x_{i,\text{miss}}^{(s)} \sim g(x_{i,\text{miss}})$;
  Generate $u \sim \mathcal{U}[0,1]$;
  Calculate the ratio $w = \frac{f(x_{i,\text{miss}}^{(s)}/g(x_{i,\text{miss}}^{(s)})}{f(x_{i,\text{miss}}^{(s-1)}/g(x_{i,\text{miss}}^{(s-1)})}$;
  if $u < w$ **then**
    Accept $x_{i,\text{miss}}^{(s)}$;
  end
  else
    $x_{i,\text{miss}}^{(s)} \leftarrow x_{i,\text{miss}}^{(s-1)}$;
  end
end
Output: $(x_{i,\text{miss}}^{(s)}, 1 \le i \le n, 1 \le s \le S)$.

3.4 Fisher 信息阵

在使用 SAEM 计算 MLE$\hat{\theta_{ML}}$ 后，我们使用 Fisher 信息矩阵 (FIM):

$$\mathcal{I}(\theta) = -\frac{\partial^2 \mathcal{LL}(\theta; x_{\text{obs}}, y)}{\partial\theta\partial\theta^T} \tag{14}$$

根据 Louis 公式，

$$\begin{aligned}\mathcal{I}(\theta) = &- \mathbb{E}\left(\frac{\partial^2 \mathcal{LL}(\theta; x, y)}{\partial\theta\partial\theta^T}\middle| x_{\text{obs}}, y; \theta\right) \\ &- \mathbb{E}\left(\frac{\partial \mathcal{LL}(\theta; x, y)}{\partial\theta}\frac{\partial \mathcal{LL}(\theta; x, y)^T}{\partial\theta}\middle| x_{\text{obs}}, y; \theta\right) \\ &+ \mathbb{E}\left(\frac{\partial \mathcal{LL}(\theta; x, y)}{\partial\theta}\middle| x_{\text{obs}}, y; \theta\right)\mathbb{E}\left(\frac{\partial \mathcal{LL}(\theta; x, y)}{\partial\theta}\middle| x_{\text{obs}}, y; \theta\right)^T\end{aligned} \tag{15}$$

因此，观测 FIM 可以用条件期望来表示，并且这些条件期望也可以通过蒙特卡洛过程来近似。更具体地说，给定从条件分布 $p(x_{i,mis}|x_{i,obs}, y; \theta_{k-1})$(7) 中抽取的缺失数据的 M 个样本 $(x_{i,mis}^{(m)}, 1 \le i \le n, 1 \le m \le M)$，观测 FIM 可以估计为

$$\hat{\mathcal{I}}_M(\hat{\theta}) = \sum_{i=1}^{n} -(D_i + G_i - \Delta_i\Delta_i^T) \tag{16}$$

其中

$$\Delta_i = \frac{1}{M}\sum_{m=1}^{M} \frac{\partial \mathcal{LL}(\hat{\theta}; x_{i,\text{miss}}^{(m)}, x_{i,\text{obs}}, y_i)}{\partial\theta} \tag{17}$$

$$D_i = \frac{1}{M}\sum_{m=1}^{M}\frac{\partial^2 \mathcal{LL}(\hat{\theta}; x_{i,\text{miss}}^{(m)}, x_{i,\text{obs}}, y_i)}{\partial\theta\partial\theta^T} \tag{18}$$

$$G_i = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{\partial \mathcal{LL}(\hat{\theta}; x_{i,\text{miss}}^{(m)}, x_{i,\text{obs}}, y_i)}{\partial\theta}\right)\left(\frac{\partial \mathcal{LL}(\hat{\theta}; x_{i,\text{miss}}^{(m)}, x_{i,\text{obs}}, y_i)}{\partial\theta}\right)^T \tag{19}$$

则具体抽样流程如下：

Algorithm 2: Calculation on observed information matrix

Input: After drawing MH samples $(x_{i,\text{miss}}^{(s)}, 1 \le i \le n, 1 \le s \le S)$ for unobserved data $(x_{i,\text{miss}}, 1 \le i \le n)$, we have imputed observations, noted as $(z_i^{(s)}, 1 \le i \le n, 1 \le s \le S)$, where $z_{ij}^{(s)} = x_{i,\text{obs}}$, if x_{ij} is observed; else $z_{ij}^{(s)} = x_{i,\text{miss}}^{(s)}$.

for $n = 1, 2, \ldots, n$ **do**

  for $s = 1, 2, \ldots, S$ **do**

    Calculate the gradient: $\nabla f_{is} = \frac{\partial \mathcal{LC}(\theta; x_{i,\text{obs}}, x_{i,\text{miss}}^{(s)}, y_i)}{\partial\beta} = z_i^{(s)}\left(y_i - \frac{\exp(\hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j z_{ij}^{(s)})}{1+\exp(\hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j z_{ij}^{(s)})}\right)$; Calculate the Hessian matrix: $H_{is} = \frac{\partial^2 \mathcal{LC}(\theta; x_{i,\text{obs}}, x_{i,\text{miss}}^{(s)}, y_i)}{\partial\partial\beta\partial\beta^T} = -z_i^{(s)} z_i^{(s)T}\frac{\exp(\hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j z_{ij}^{(s)})}{(1+\exp(\hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j z_{ij}^{(s)})}^2$;

    $\Delta_i \leftarrow \frac{1}{s}[(s-1)]\Delta_i + \nabla f_{is}]$;

    $D_i \leftarrow \frac{1}{s}[(s-1)D_i + H_{is}]$;

    $G_i \leftarrow \frac{1}{s}[(s-1)G_i + \nabla f_{is}\nabla f_{is}^T]$;

  end

end

Output: $\hat{\mathcal{I}}_S(\hat{\beta}) \leftarrow \hat{\mathcal{I}}_S(\hat{\beta}) - (D_i + G_i - \Delta_i\Delta_i^T)$;

3.5 模型选择

为了比较不同的模型，我们可以考虑使用惩罚似然准则，例如 AIC,BIC 准则，对于给定的模型 M 和参数 $\hat{\theta_M}$, 其定义如下:

$$\text{AIC}(\mathcal{M}) = -2\mathcal{LL}(\hat{\theta}_{\mathcal{M}}; x_{\text{obs}}, y) + 2d(\mathcal{M}) \tag{20}$$

$$\text{BIC}(\mathcal{M}) = -2\mathcal{LL}(\hat{\theta}_{\mathcal{M}}; x_{\text{obs}}, y) + \log(n)d(\mathcal{M}) \tag{21}$$

其中 $d(M)$ 为模型 M 中估计参数的数量，$x = x_i j, 1 \le i \le n, 1 \le j \le n)$ 的分布并不依赖于回归模型，故而模型之间 $d(M)$ 的差异等同于 β_M 中非零系数的差异。
对于给定的模型与参数 θ, 观测对数似然函数定义为

$$\mathcal{LL}(\theta; x_{\text{obs}}, y) = \sum_{i=1}^{n}\log\left(\mathbf{p}(y_i, x_{i,\text{obs}}; \theta)\right) \tag{22}$$

对于每一个 i,$p(x_{i,obs}, y; \theta)$ 无法显示表示，同时为了缩减估计量的方差，我们可以采用蒙特卡洛方法中的重要抽要法来近似估计，设 g_i 为 $x_{i,\text{miss}}|x_{i,\text{obs}} \sim \mathcal{N}_p(\mu_i, \Sigma_i)$ 的正态密度函数，则

$$\begin{aligned}
\mathbf{p}(y_i, x_{i,\text{obs}}; \theta) &= \int \mathbf{p}(y_i, x_{i,\text{obs}} \mid x_{i,\text{miss}}; \theta)\mathbf{p}(x_{i,\text{miss}}; \theta)dx_{i,\text{miss}} \\
&= \int \mathbf{p}(y_i, x_{i,\text{obs}} \mid x_{i,\text{miss}}; \theta)\frac{\mathbf{p}(x_{i,\text{miss}}; \theta)}{g_i(x_{i,\text{miss}})}g_i(x_{i,\text{miss}})dx_{i,\text{miss}} \\
&= \mathbb{E}_{g_i}\left(\mathbf{p}(y_i, x_{i,\text{obs}} \mid x_{i,\text{miss}}; \theta)\frac{\mathbf{p}(x_{i,\text{miss}}; \theta)}{g_i(x_{i,\text{miss}})}\right)
\end{aligned} \tag{23}$$

所以当我们从建议分布中抽取 M 个样本:$x_{i,\text{miss}}^{(m)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_i, \Sigma_i), \quad m = 1, 2, \cdots, M$ 我们可以通过

$$\hat{p}(y_i, x_{i,\text{obs}}; \theta) = \frac{1}{M}\sum_{m=1}^{M}\mathbf{p}(y_i, x_{i,\text{obs}} \mid x_{i,\text{miss}}^{(m)}; \theta)\frac{\mathbf{p}(x_{i,\text{miss}}^{(m)}; \theta)}{g_i(x_{i,\text{miss}}^{(m)})} \tag{24}$$

来估计 $\hat{p}(y_i, x_{i,obs}; \theta)$ 从而计算观测对数似然函数 $\mathcal{LL}(\theta; x_{\text{obs}}, y)$

4 模拟实验

4.1 SAEM 的收敛性

我们首先生成了一个大小为 $(n = 1000) \times (p = 5)$ 的设计矩阵 x，x_{ij} 从多元正态分布 $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ 中抽取观测值来生成。然后，我们根据逻辑回归模型 (1) 生成了响应变量。我们考虑的真实参数值为：$\boldsymbol{\beta} = (-0.2, 0.5, -0.3, 1, 0, -0.6), \quad \boldsymbol{\mu} = (1, 2, 3, 4, 5), \quad \boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma})\mathbf{C}\text{diag}(\boldsymbol{\sigma})$, 其中 $\boldsymbol{\sigma}$ 是标准差向量 $\boldsymbol{\sigma} = (1, 2, 3, 4, 5)$，$\mathbf{C}$ 是相关矩阵:

$$C = \begin{bmatrix} 1 & 0.8 & 0 & 0 & 0 \\ 0.8 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.3 & 0.6 \\ 0 & 0 & 0.3 & 1 & 0.7 \\ 0 & 0 & 0.6 & 0.7 & 1 \end{bmatrix} \tag{25}$$

数据生成完毕以后，我们在其中引入 10% 的随机数据缺失, 我们首先使用完全随机缺失 (MCAR) 机制, 其中每个变量被观测到的概率相同, 详细代码见附录。其中参数的初始化选择使用均值插补，即用每个变量在其观测值上的均值填补缺失值，在前 k_1 次迭代中，我们选择 $\gamma_k = 1$, 使之快速的收敛到最大似然估计附近，而在第 k_1 次迭代之后，令 $\gamma_k = (k - k_1)^{-\tau}$ 使该算法几乎必然收敛。之后为了研究步长序列 γ_k 和 τ 对算法收敛性的影响，我们固定 $k_1 = 50$, 并分别使用 $\tau = (0.6, 0.8, 1)$ 作为迭代算法的步长，详细代码请见附录。我们先不妨只对 β_1 的收敛情况进行分析，结果如下：

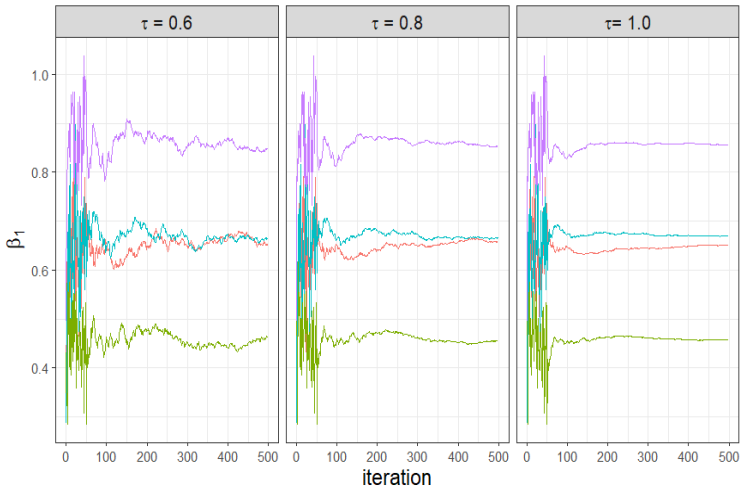

Figure 1: 在不同步长 τ 下 β_1 的收敛图像，每一种颜色对应着一次随机种子的模拟

图一展示了 SAEM 算法在四个不同种子的模拟数据集下系数 β_1 收敛情况，从图中可以看到，对于相同随机种子下的数值模拟，三个估计序列都收敛到相同的解，并且对于较大的 τ，SAEM 收敛地更快，波动更小。所以我们不妨在后续的数值模拟中取 $\gamma = 1$。SAEM 算法在估计 β 的其他分量时结果类似：

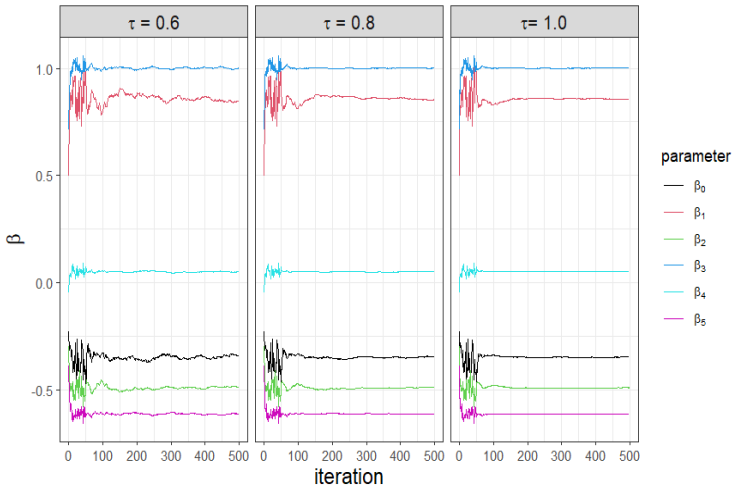

Figure 2: 在不同步长 τ 下，β 各分量的收敛情况

4.2 SAEM 算法与其他算法参数覆盖率的比较

为了探究 DAEM 算法与其他算法的差异，我们生成了来自于标准正态的 200×3 的协变量矩阵，真实参数首元素为 1，其余 p 个为 0.5，随机导入缺失率为 0.2 的缺失数据，分别使用 SAEM，MICE,Amelica 算法估计其参数，偏差、均方根误差和置信区间覆盖率，得到如下结果：

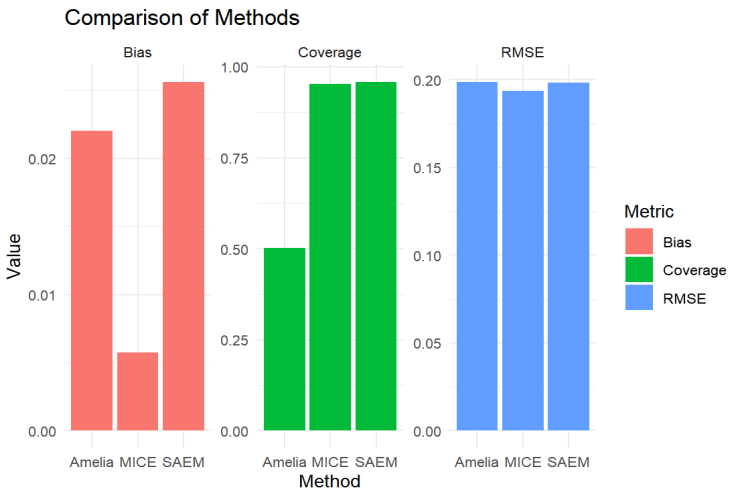

Figure 3: 不同算法下参数的偏差，均方根误差与覆盖率

结果表明 MICE 方法的偏差与均方根误差最小，SAEM 算法的覆盖率最大，可以知道 SAEM 算法再参数覆盖层面具有良好性能。

4.3 SAEM 在不同缺失机制与缺失率下的表现

4.3.1 MAR 与 MCAR

MCAR 完全随机缺失: 数据的缺失与任何值都无关，即 $\mathbf{p}(M_i \mid y, x_i, \phi) = \mathbf{p}(M_i \mid \phi)$ MAR 随机缺失：数据缺失的概率可能依赖于观测到的数据，但不依赖于缺失的数据，详细定义如下，通过将数据分解为 x_i 分解为一个可能缺失的子集 x_i^{miss}, 和一个不可能缺失的子集 x_i^{obs}(即总被观测到的数据)，则观测到的数据 $x_{i,miss}$ 必然包括 x_i^{miss}, 可能缺失的数据 x_i^{miss} 必然包括缺失的数据 x_i^{miss}, 故而 MAR 意味着对任意 i,

$$\mathbf{p}(r_i|y_i, x_i; \phi) = \mathbf{p}(r_i|y_i, x_{(\text{obs})}; \phi) = \mathbf{p}(r_i|y_i, x_{i,\text{obs}}; \phi) \tag{26}$$

所以 MAR 意味着观测似然函数可以被最大化，同时 r 的分布可以被忽略：

$$
\begin{aligned}
\mathcal{L}(\theta, \phi; y, x_{\text{obs}}, r) &= \mathbf{p}(y, x_{\text{obs}}, r; \theta, \phi) \\
&= \prod_{i=1}^{n} \mathbf{p}(y_i, x_{i,\text{obs}}, r_i; \theta, \phi) \\
&= \prod_{i=1}^{n} \int \mathbf{p}(y_i, x_i, r_i; \theta, \phi) dx_{i,\text{miss}} \\
&= \prod_{i=1}^{n} \int \mathbf{p}(y_i, x_i; \theta) \mathbf{p}(r_i | y_i, x_i; \phi) dx_{i,\text{miss}} \\
&= \prod_{i=1}^{n} \int \mathbf{p}(y_i, x_i; \theta) \mathbf{p}(r_i | y_i, x_{i,\text{obs}}; \phi) dx_{i,\text{obs}} \\
&= \prod_{i=1}^{n} \mathbf{p}(r_i | y_i, x_{i,\text{obs}}; \phi) \times \prod_{i=1}^{n} \int \mathbf{p}(y_i, x_i; \theta) dx_{i,\text{miss}} \\
&= \mathbf{p}(r | y, x_{\text{obs}}; \phi) \times \mathbf{p}(y, x_{\text{obs}}; \theta) \\
&= \mathbf{p}(r | y, x^{(\text{obs})}; \phi) \times \mathbf{p}(y, x_{\text{obs}}; \theta)
\end{aligned}
\tag{27}
$$

故而，求参数的 MLE 即等价为最大化 $\mathbf{p}(y, x_{\text{obs}}; \theta)$

4.3.2 算法结果

为了探究 SAEM 算法在不同缺失机制下的表现，我们在 R 中数据 mtcars 中使用 MAR 与 MCAR 两种数据缺失机制，和 0.05，0.10，0.15 三种缺失率，利用 SAEM 算法判断估计参数的 MSE 与 MAE，结果如下：

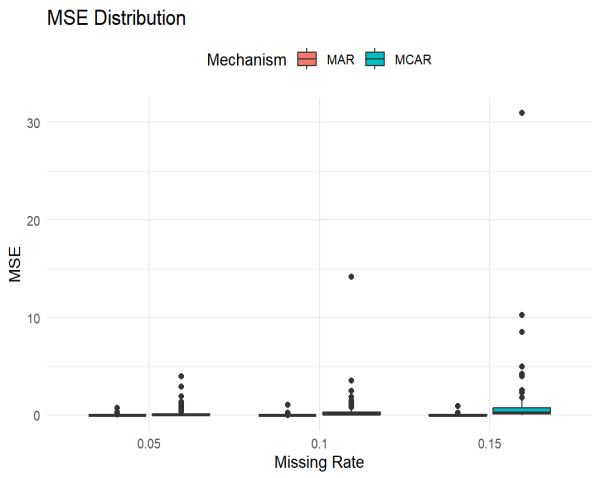

Figure 4: MSE

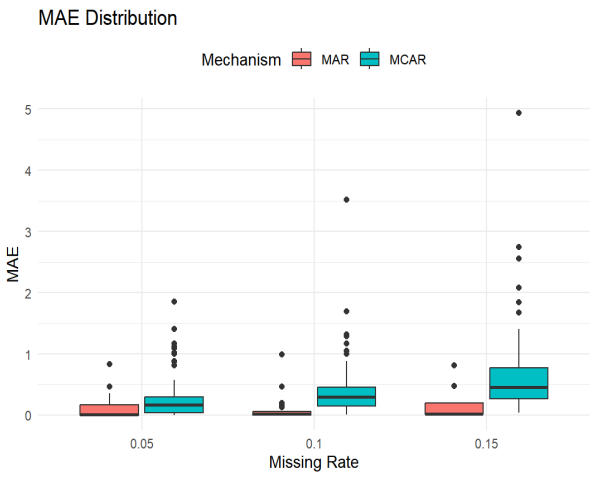

Figure 5: MAE

通过上述结果我们可以发现 SAEM 方法在 MCAR 与 MAR 数据缺失机制下都有较好表现，数据缺失率越低其 MSE 与 MAE 值越小，符合常识，并且在缺失率相同条件下，SAEM 在处理 MAR 缺失问题时 MSE 与 MAE 较小，SAEM 方法效果更好。

4.4 变量选择

为考察该方法在变量选择方面的能力，我们沿用与第一个数值实验相同的模拟场景，但将一些参数设置为零。我们现在描述所有 β 中的参数都为零，除了 $\beta_0 = -0.2$ $\beta_1 = 0.5$ $\beta_3 = 1$ $\beta_5 = -0.6$ 的情况的结果。我们考虑基于惩罚似然的标准，AIC 和 BIC，来进行变量选择，之后，对于每一种变量组合，我们使用 SAEM 估计参数，然后计算其观测对数似然从而求得 AIC 与 BIC 值。最后，我们根据 AIC 或 BIC 的最小值选择最佳模型。我们利用 R 进行了 100 次模拟实验，下图为在前十次模拟中 AIC 与 BIC 值的变化, 可以发现 AIC 与 BIC 值先迅速减小，再趋于平稳。

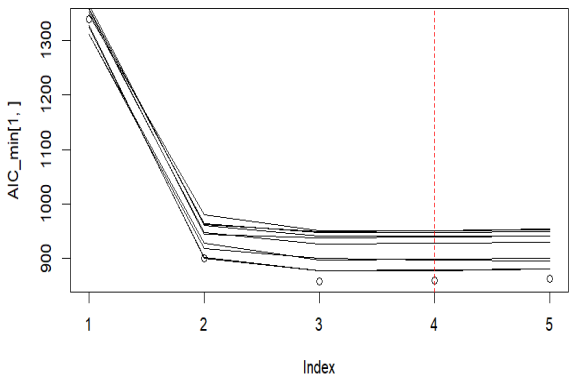

Figure 6: AIC

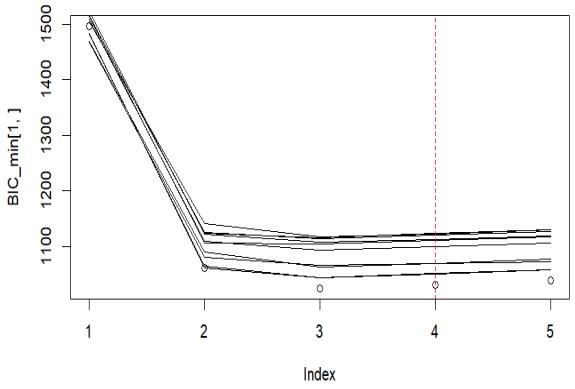

Figure 7: BIC

之后为了判断 AIC 与 BIC 进行模型选择的优劣，我们计算了以下百分比：每个标准选择真实模型（C）的百分比，过度拟合（O）的百分比 (即选择的变量比实际多的情况)，以及欠拟合（U）的百分比 (即选择的变量比实际少的情况), 得到下表：

Model Selection	**Percentage of C**	**Percentage of O**	**Percentage of U**
AIC	68%	32%	0%
BIC	99%	1%	0%

Table 1: Model Selection Results

结果显示，使用 AIC 进行变量选择时，过度拟合模型的可能性较大，而 BIC 的结果则更好。因此，我们可以使用 BIC 来进行模型选择。

5 总结

EM 算法一直被视为处理缺失值的自然解决方案，因为它在 MAR 缺失机制中提供了最大似然估计，但传统 EM 算法有时难以直接计算 E 步中的期望，而 MCEM 与 SAEM 为之提供了一个很好的解决方案。在本文中，我们研究了一个处理带有缺失值的逻辑回归问题，提出 SAEM 算法，并利用蒙特卡罗方法研究其收敛性，参数估计的性能，在不同缺失机制下的表现以及变量选择问题，并发现，我们的方法计算效率高，易于实现，参数覆盖率高。基于我们提出的算法，可以自然地使用带有缺失数据的 BIC 进行变量选择。

A 附录：代码

Example:

```
library(misaem)
library(MASS)
library(mvtnorm)
library(ggplot2)
library(reshape2)
library(dplyr)
library(RColorBrewer)
theme_set(theme_bw())
```

miss.saem函数并没有包含在作者的misaem包里，作者在前几次更新中将其删去。
我们团队致力于根据misaem里的新函数和作者的理论思路将miss.saem函数复现了出来，
这是整个项目里难度最大的地方。

```
miss.saem <- function(X.obs,y,pos_var=1:ncol(X.obs),maxruns=500,tol_em=1e-7,nmcmc=2
  set.seed(seed)

  #judge
  #if (class(X.obs) == "data.frame") {
  #  X.obs <- as.matrix(X.obs)
  #}
  if (sum(sapply(X.obs, is.numeric)) < ncol(X.obs)) {
    stop("Error: the variables should be numeric.")
  }
  if (sum(y==1) +  sum(y==0) < nrow(X.obs)) {
    stop("Error: y must be coded by 0 or 1, and there is no missing data in y.")
  }

  if (sum(pos_var %in% 1:ncol(X.obs)) < length(pos_var))  {
    stop("Error: index of selected variables must be in the range of covariates.")
  }

  if (length(unique(pos_var)) != length(pos_var)){
    stop("Error: index of selected variables must not be repeated.")
  }

  p=ncol(X.obs)
```

```
#delete rows completely missing
if(any(apply(is.na(X.obs),1,sum)==p)){
  i_allNA=which(apply(is.na(X.obs),1,sum)==p)
  X.obs = X.obs[-i_allNA,]
  y = y[-i_allNA]
}
if(any((is.na(y))==TRUE)){
  i_YNA=which(is.na(y)==TRUE)
  X.obs = X.obs[-i_YNA,]
  y = y[-i_YNA]
}
n=length(y)


rindic = as.matrix(is.na(X.obs))
if(sum(rindic)>0){
  whichcolmissing = (1:ncol(rindic))[apply(rindic,2,sum)>0]
  missingcols = length(whichcolmissing)
}
if(sum(rindic)==0){missingcols=0}


ptm <- Sys.time()
if(missingcols>0){
  k=0
  cstop=0.1
  seqbeta = matrix(NA,nrow=ncol(X.obs)+1,ncol=(maxruns+1))
  seqbeta_avg = matrix(NA,nrow=ncol(X.obs)+1,ncol=(maxruns+1))

  X.mean = X.obs
  for(i in 1:ncol(X.mean)){
    X.mean[is.na(X.mean[,i]), i] <- mean(X.mean[,i], na.rm = TRUE)
  }
  X.sim <- X.mean

  mu = apply(X.mean,2,mean)
  Sigma = var(X.mean)*(n-1)/n
  beta= rep(0,p+1)
```

```
beta[c(1,pos_var+1)]= glm(y~ X.mean[,pos_var],family=binomial(link='logit'))$co

while ((cstop>tol_em)*(k<maxruns)|(k<20)){
  k = k+1
  beta.old = beta

  if(k <k1){gamma <- 1}else{gamma <- 1/(k-(k1-1))^tau}

  S.inv <- solve(Sigma)

  for (i in (1:n)) {
    jna <- which(is.na(X.obs[i,]))
    njna <- length(jna)
    if (njna>0) {
      xi <- X.sim[i,]
      Oi <- solve(S.inv[jna,jna])
      mi <- mu[jna]
      lobs <- beta[1]
      if (njna<p) {
        jobs <- setdiff(1:p,jna)
        mi <- mi - (xi[jobs] - mu[jobs])%*%S.inv[jobs,jna]%*%Oi
        lobs <- lobs + sum(xi[jobs]*beta[jobs+1])
      }

      cobs <- exp(lobs)
      if(cobs==0){cobs=.Machine$double.xmin}
      if(cobs==Inf){cobs=.Machine$double.xmax}

      xina <- xi[jna]
      betana <- beta[jna+1]
      for (m in (1:nmcmc)) {
        xina.c <- mi + rnorm(njna)%*%chol(Oi)

        if (y[i]==1)
          alpha <- (1+exp(-sum(xina*betana))/cobs)/(1+exp(-sum(xina.c*betana))/
        else
          alpha <- (1+exp(sum(xina*betana))*cobs)/(1+exp(sum(xina.c*betana))*co
        if (runif(1) < alpha){
          xina <- xina.c
```

```
        }
      }
      X.sim[i,jna] <- xina
    }
  }
  beta_new= rep(0,p+1)
  beta_new[c(1,pos_var+1)]= glm(y~ X.sim[,pos_var],family=binomial(link='logit'

  beta <- (1-gamma)*beta + gamma*beta_new
  cstop = sum((beta-beta.old)^2)

  mu <- (1-gamma)*mu + gamma*colMeans(X.sim)
  Sigma <- (1-gamma)*Sigma + gamma*cov(X.sim)

  seqbeta[,k]=beta.old

  if(k==1){
    seqbeta_avg[,k]=beta.old
  }else{
    seqbeta_avg[,k]= 1/k*rowSums(seqbeta[,1:k])
  }

  if(print_iter==TRUE & k %% 10 == 0){
    cat(sprintf('iteration = %i ', k))
    cat(sprintf('beta ='),beta,'\n')
    cat(sprintf('Distance from last iteration ='),cstop,'\n')
  }
 }
 var_obs = ll = std_obs =NULL
 if(var_cal==TRUE){
  var_obs = louis_lr_saem(beta,mu,Sigma,y,X.obs,pos_var,rindic,whichcolmissing,
  std_obs <- sqrt(diag(var_obs))
 }
 if(ll_obs_cal==TRUE){
  ll= likelihood_saem(beta,mu,Sigma,y,X.obs,rindic,whichcolmissing,mc.size=1000
 }
}
if(missingcols==0){
 X.obs = matrix(X.obs,nrow=n)
```

```
    data.complete <- data.frame(y=y,X.obs)
    model.complete <- glm(y ~.,family=binomial(link='logit'),data=data.complete)
    mu = apply(X.obs,2,mean)
    Sigma = var(X.obs)*(n-1)/n
    beta <- model.complete$coefficients
    var_obs = ll = ll1 =ll2= std_obs =seqbeta_avg= seqbeta=NULL
    if(var_cal==TRUE){
      P <- predict(model.complete, type = "response")
      W <- diag(P*(1-P))
      X <- model.matrix(model.complete)

      var_obs <- solve(t(X)%*%W%*%X)
      std_obs <- sqrt(diag(var_obs))
    }
    if(ll_obs_cal==TRUE){
      ll = likelihood_saem(beta,mu,Sigma,y,X.obs,rindic,whichcolmissing,mc.size=100
    }
  }
  time_run=Sys.time() - ptm
  return(list(mu=mu, sig2=Sigma, beta=beta,time_run=time_run,seqbeta=seqbeta,seqbet
}
```

我们首先生成一个大小为 n=1000 乘以 p=5 的设计矩阵，通过从多元正态分布 N(,Σ) 中抽取

```{r}
n <- 1000  # number of subjects
p <- 5     # number of explanatory variables
mu.star <- 1:p  # mean of the explanatory variables
sd <- 1:p # standard deviations
C <- matrix(c(   # correlation matrix
  1,   0.8, 0,   0,   0,
  0.8, 1,   0,   0,   0,
  0,   0,   1,   0.3, 0.6,
  0,   0,   0.3, 1,   0.7,
  0,   0,   0.6, 0.7, 1
), nrow=p)
Sigma.star <- diag(sd)%*%C%*%diag(sd) # variance-covariance matrix of the explanato

beta.star <- c(0.5, -0.3, 1, 0, -0.6) # coefficients of logistic regression
beta0.star <- -0.2  # intercept
```

```
beta.true = c(beta0.star,beta.star)

# generate complete design matrix
X.complete <- matrix(rnorm(n*p), nrow=n)%*%chol(Sigma.star) + matrix(rep(mu.star,n)
# generate response vector
p1 <- 1/(1+exp(-X.complete%*%beta.star-beta0.star))
y <- as.numeric(runif(n)<p1)
```

然后，我们根据完全随机缺失（MCAR）机制，在协变量中随机引入10%的缺失值

```{r}
p.miss <- 0.10
patterns = runif(n*p)<p.miss
X.obs <- X.complete
X.obs[patterns] <- NA
list.saem=miss.saem(X.obs,y,print_iter = FALSE,var_cal = TRUE, ll_obs_cal = TRUE)
cat("Estimated beta: ", '\n', list.saem$beta, '\n')
cat("Variance-covariance matrix for estimation: ", '\n', list.saem$var_obs, '\n')
cat("Standard error for estimation: ", '\n', list.saem$std_obs, '\n')
cat("Observed log-likelihood: ", '\n', list.saem$ll, '\n')
cat("Execution time: ", '\n', list.saem$time_run, '\n')
```

Convergence of SAEM

为了研究SAEM（随机近似EM算法）关于步长 γ_k 的收敛性，我们选择在前 k_1 次迭

```{r}
NB = 4 # number of repetitions of simulations
tau <- c(0.6, 0.8, 1)
k1 <- 50
maxruns=500
BIASBETA1_0.6 = BETA1_0.6 = matrix(0, NB, maxruns+1)
BIASBETA1_0.8 = BETA1_0.8 = matrix(0, NB, maxruns+1)
BIASBETA1_1.0 = BETA1_1.0 = matrix(0, NB, maxruns+1)

seed <- c(1,100,1000,10000)

for(nb in 1:NB){
```

```
  set.seed(seed[nb])
  # ----- complete data
  X.complete <- matrix(rnorm(n*p), nrow=n)%*%chol(Sigma.star) + matrix(rep(mu.star,
  p1 <- 1/(1+exp(-X.complete%*%beta.star-beta0.star))
  y <- as.numeric(runif(n)<p1)

  # ------- generating missing data
  X.obs <- X.complete
  patterns = runif(n*p)<p.miss
  X.obs[patterns] <- NA

  # tau = 0.6
  list.saem0.6=miss.saem(X.obs,y,maxruns=maxruns,tol_em=1e-50,tau=tau[1],k1=k1,prin
  BETA1_0.6[nb,] = list.saem0.6$seqbeta[2,]
  BIASBETA1_0.6[nb,] = list.saem0.6$seqbeta[2,] - list.saem0.6$beta[2]

  # tau = 0.8
  list.saem0.8=miss.saem(X.obs,y,maxruns=maxruns,tol_em=1e-50,tau=tau[2],k1=k1,prin
  BETA1_0.8[nb,] = list.saem0.8$seqbeta[2,]
  BIASBETA1_0.8[nb,] = list.saem0.8$seqbeta[2,] - list.saem0.8$beta[2]

  # tau = 1.0
  list.saem1.0=miss.saem(X.obs,y,maxruns=maxruns,tol_em=1e-50,tau=tau[3],k1=k1,prin
  BETA1_1.0[nb,] = list.saem1.0$seqbeta[2,]
  BIASBETA1_1.0[nb,] = list.saem1.0$seqbeta[2,] - list.saem1.0$beta[2]
}
```

这里我们生成收敛图。 使用三个不同值的 τ 得到的 β_1 的收敛图。每种颜色代表

```{r}
# pdf('saem_gammak.pdf',width = 11, height = 8 ,onefile = T) # save as pdf
fnames <- c("0.6", "0.8", "1.0")
df1 <- as.data.frame(t(BETA1_0.6))
names(df1) <- 1:NB
df1['iteration'] <- 0:(nrow(df1)-1)
df1 <- melt(df1, variable.name="replicate", id.vars = list("iteration"))
df1['tau'] = fnames[1]
df2 <- as.data.frame(t(BETA1_0.8))
```

```
names(df2) <- 1:NB
df2['iteration'] <- 0:(nrow(df2)-1)
df2 <- melt(df2, variable.name="replicate", id.vars = list("iteration"))
df2['tau'] = fnames[2]
df3 <- as.data.frame(t(BETA1_1.0))
names(df3) <- 1:NB
df3['iteration'] <- 0:(nrow(df3)-1)
df3 <- melt(df3, variable.name="replicate", id.vars = list("iteration"))
df3['tau'] = fnames[3]

df <- rbind(df1, df2, df3)
df[['tau']] <- factor(df[['tau']], levels=fnames)
levels(df[['tau']]) <- c("tau*' = 0.6'", "tau*' = 0.8'", "tau*'= 1.0'")

beta2 <- subset(df, iteration==maxruns)
beta1 <- beta2
beta1$iteration <- 0
beta <- rbind(beta1, beta2)

pl <- ggplot(df) + geom_line(aes(iteration,value,color=replicate)) +
  geom_line(data=beta, aes(iteration, value, color=replicate), linetype=3) +
  facet_grid(~tau, labeller = label_parsed) +  ylab(expression(beta[1])) +
  theme(strip.text = element_text(size=12), axis.title=element_text(size=14),
        legend.position="none")
print(pl)
```

SAEM中所有 β 的收敛图。每种颜色代表一个参数：

```{r}
# pdf('converge_tau_all_beta.pdf',width = 11, height = 8 ,onefile = T) # save as pd
df1 <- as.data.frame(t(list.saem0.6$seqbeta))
names(df1) <- paste0("beta[",1:6,"]")
df1['iteration'] <- 0:(nrow(df1)-1)
df1 <- melt(df1, variable.name="parameter", id.vars = list("iteration"))
df1['tau'] = fnames[1]
df2 <- as.data.frame(t(list.saem0.8$seqbeta))
names(df2) <- paste0("beta[",1:6,"]")
df2['iteration'] <- 0:(nrow(df2)-1)
```

```
df2 <- melt(df2, variable.name="parameter", id.vars = list("iteration"))
df2['tau'] = fnames[2]
df3 <- as.data.frame(t(list.saem1.0$seqbeta))
names(df3) <- paste0("beta[",1:6,"]")
df3['iteration'] <- 0:(nrow(df3)-1)
df3 <- melt(df3, variable.name="parameter", id.vars = list("iteration"))
df3['tau'] = fnames[3]

df <- rbind(df1, df2, df3)
df[['tau']] <- factor(df[['tau']], levels=fnames)
levels(df[['tau']]) <- c("tau*' = 0.6'", "tau*' = 0.8'", "tau*'= 1.0'")

beta2 <- subset(df, iteration==maxruns)
beta1 <- beta2
beta1$iteration <- 0
beta <- rbind(beta1, beta2)

ldf <- levels(df$parameter)
labl <- list(expression(beta[0]), expression(beta[1]), expression(beta[2]),
             expression(beta[3]), expression(beta[4]), expression(beta[5]) )

palette(brewer.pal(6, "Dark2"))
pl <- ggplot(df) + geom_line(aes(iteration,value,color=parameter)) +
#  geom_line(data=beta, aes(iteration, value, color=replicate)) +
  facet_grid(~tau, labeller = label_parsed) +  ylab(expression(beta)) +
  scale_color_manual(labels = labl, values=1:6) +
  theme(strip.text = element_text(size=12), axis.title=element_text(size=14))
print(pl)
```

在使用SAEM估计参数之后，下一个目标是在存在缺失值的情况下进行变量选择。

```{r}
library(misaem)
library(MASS)
library(mvtnorm)
```

这里我们首先给定参数的真实值。（通过使用不同的值，我们可以构建不同的设置，例如受试

```{r}
```

```r
n <- 1000  # number of subjects
p <- 5     # number of explanatory variables
mu.star <- 1:p  # mean of the explanatory variables
sd <- 1:p # standard deviations

# with correlation
C <- matrix(c(   # correlation matrix
  1,   0.8, 0,   0,   0,
  0.8, 1,   0,   0,   0,
  0,   0,   1,   0.3, 0.6,
  0,   0,   0.3, 1,   0.7,
  0,   0,   0.6, 0.7, 1
), nrow=p)
## or without correlation
# C = diag(p)

Sigma.star <- diag(sd)%*%C%*%diag(sd) # variance-covariance matrix of the explanato

beta.star <- c(0.5, 0, 1, 0, -0.6)  # coefficients of logistic regression
beta0.star <- -0.2  # intercept
beta.true = c(beta0.star,beta.star)

#percentage of missingness
p.miss <- 0.10
```

我们考虑基于惩罚似然的标准，如AIC和BIC，来进行变量选择。\
对于每一种变量组合，我们使用SAEM估计参数，然后计算观测对数似然。最后，我们根据AIC或
我们进行了100次模拟重复，并计算了以下百分比：每个标准选择真实模型（C）的百分比，过

```r
nb.simu = 100

subsets=combinations(p)

ll = AIC = BIC = matrix(0, nrow = nb.simu, ncol = nrow(subsets)-1)

AIC_min =BIC_min = matrix(1e+5,nrow = nb.simu,ncol = p)
j_AIC = j_BIC  = matrix(0,nrow = nb.simu,ncol = p)
```

```
AIC_all_min =BIC_all_min = rep(1e+5,nb.simu)
j_all_AIC = j_all_BIC = rep(0,nb.simu)

for(nb in 1:nb.simu){
  set.seed(nb)
  cat('simu ',nb,'\n')
  # complete data simulation
  X.complete <- matrix(rnorm(n*p), nrow=n)%*%chol(Sigma.star) + matrix(rep(mu.star,
  p1 <- 1/(1+exp(-X.complete%*%beta.star-beta0.star))
  y <- as.numeric(runif(n)<p1)

  # generate missingness
  X.obs <- X.complete
  patterns = runif(n*p)<p.miss
  X.obs[patterns] <- NA

  # iterate among each combination
  for (j in 1:(nrow(subsets)-1)){
    nb.var = sum(subsets[j,])
    variables = subsets[j,]
    pos_var=which(variables==1)
    nb.x = sum(variables)
    nb.para = (nb.x + 1) + p + p*p
    list.saem.subset=miss.saem(X.obs,y,pos_var,maxruns=1000,tol_em=1e-7,nmcmc=2,tau
    ll[nb,j] = list.saem.subset$ll
    AIC[nb,j] = -2*ll[nb,j]+ 2*nb.para
    BIC[nb,j] = -2*ll[nb,j]+ nb.para * log(n)

    if(AIC[nb,j]<=AIC_min[nb,nb.x]){
      AIC_min[nb,nb.x]= AIC[nb,j]
      j_AIC[nb,nb.x] = j
    }
    if(BIC[nb,j]<=BIC_min[nb,nb.x]){
      BIC_min[nb,nb.x]= BIC[nb,j]
      j_BIC[nb,nb.x] = j
    }
    if(AIC[nb,j]<=AIC_all_min[nb]){
      AIC_all_min[nb]= AIC[nb,j]
```

```
      j_all_AIC[nb] = j
    }
    if(BIC[nb,j]<=BIC_all_min[nb]){
      BIC_all_min[nb]= BIC[nb,j]
      j_all_BIC[nb] = j
    }
  }
}

# 初始化计数器
count_C_AIC = count_O_AIC = count_U_AIC = 0
count_C_BIC = count_O_BIC = count_U_BIC = 0

# 遍历每次模拟结果
for (nb in 1:nb.simu) {
  # 根据 AIC 选择的最佳模型
  selected_AIC_model = subsets[j_all_AIC[nb], ]
  diff_AIC = sum(selected_AIC_model) - sum(beta.star != 0)

  if (diff_AIC == 0) {
    count_C_AIC = count_C_AIC + 1  # 真实模型
  } else if (diff_AIC > 0) {
    count_O_AIC = count_O_AIC + 1  # 过度拟合
  } else {
    count_U_AIC = count_U_AIC + 1  # 欠拟合
  }

  # 根据 BIC 选择的最佳模型
  selected_BIC_model = subsets[j_all_BIC[nb], ]
  diff_BIC = sum(selected_BIC_model) - sum(beta.star != 0)

  if (diff_BIC == 0) {
    count_C_BIC = count_C_BIC + 1  # 真实模型
  } else if (diff_BIC > 0) {
    count_O_BIC = count_O_BIC + 1  # 过度拟合
  } else {
    count_U_BIC = count_U_BIC + 1  # 欠拟合
  }
}
```

```
# 计算百分比
percent_C_AIC = 100 * count_C_AIC / nb.simu
percent_O_AIC = 100 * count_O_AIC / nb.simu
percent_U_AIC = 100 * count_U_AIC / nb.simu

percent_C_BIC = 100 * count_C_BIC / nb.simu
percent_O_BIC = 100 * count_O_BIC / nb.simu
percent_U_BIC = 100 * count_U_BIC / nb.simu

# 输出结果
cat("AIC Model Selection:\n")
cat("Percentage of Correct (C):", percent_C_AIC, "%\n")
cat("Percentage of Overfitting (O):", percent_O_AIC, "%\n")
cat("Percentage of Underfitting (U):", percent_U_AIC, "%\n")

cat("\nBIC Model Selection:\n")
cat("Percentage of Correct (C):", percent_C_BIC, "%\n")
cat("Percentage of Overfitting (O):", percent_O_BIC, "%\n")
cat("Percentage of Underfitting (U):", percent_U_BIC, "%\n")
```

绘制几次模拟的BIC或AIC图。

```{r}
plot(AIC_min[1,])
for (i in 1:10){lines(AIC_min[i+1,])}
abline(v = 4, col = "red", lty = 2)

plot(BIC_min[1,])
for (i in 1:10){lines(BIC_min[i+1,])}
abline(v = 4, col = "red", lty = 2)
```

我们现在想要评估SAEM算法的性能

```{r}
library(misaem)
library(MASS)
```

```
library(mvtnorm)
library(ggplot2)
library(RColorBrewer)
theme_set(theme_bw())
library(tidyr)
library(dplyr)
library(xtable)
```

评估saem性能

```{r}
# 加载必要的包
library(ggplot2)
library(tidyr)
library(dplyr)
library(xtable)

# 修改MCAR和MAR的数据生成函数，确保生成的缺失值模式更稳定
create_MCAR <- function(X, missing_rate) {
  X_miss <- as.matrix(X)
  n <- nrow(X)
  p <- ncol(X)
  for(j in 1:p) {
    missing_indices <- rbinom(n, 1, missing_rate)
    X_miss[missing_indices == 1, j] <- NA
  }
  return(X_miss)
}

create_MAR <- function(X, missing_rate) {
  X_miss <- as.matrix(X)
  n <- nrow(X)
  p <- ncol(X)

  # 使用第一个变量的值来生成缺失概率
  prob_miss <- pnorm(scale(X[,1]))
  prob_miss[is.na(prob_miss)] <- mean(prob_miss, na.rm = TRUE)
```

```
  for(j in 2:p) {
    missing_probs <- pmin(prob_miss * missing_rate, 1)
    missing_indices <- rbinom(n, 1, missing_probs)
    X_miss[missing_indices == 1, j] <- NA
  }
  return(X_miss)
}

# 修改测试函数，添加错误处理
run_test <- function(X, y, mechanism, rate, beta_complete) {
  tryCatch({
    start_time <- Sys.time()

    # 创建缺失数据
    if(mechanism == "MCAR") {
      X_miss <- create_MCAR(X, rate)
    } else {
      X_miss <- create_MAR(X, rate)
    }

    # 检查缺失值比例
    actual_missing_rate <- sum(is.na(X_miss)) / (nrow(X_miss) * ncol(X_miss))
    if(abs(actual_missing_rate - rate) > 0.1) {
      warning(sprintf("实际缺失率 (%f) 与目标缺失率 (%f) 差异过大",
                      actual_missing_rate, rate))
    }

    # 运行SAEM
    saem_result <- miss.saem(X_miss, y, pos_var=1:ncol(X),
                             maxruns=500, tol_em=1e-6,
                             print_iter=FALSE, var_cal=TRUE)

    end_time <- Sys.time()
    runtime <- as.numeric(difftime(end_time, start_time, units = "secs"))

    # 计算性能指标
    mse <- mean((saem_result$beta - beta_complete)^2, na.rm = TRUE)
    mae <- mean(abs(saem_result$beta - beta_complete), na.rm = TRUE)
```

```
    return(list(
      beta = saem_result$beta,
      std_obs = saem_result$std_obs,
      mse = mse,
      mae = mae,
      runtime = runtime,
      success = TRUE
    ))
  }, error = function(e) {
    warning(sprintf("实验失败: %s", e$message))
    return(list(
      beta = rep(NA, length(beta_complete)),
      std_obs = NA,
      mse = NA,
      mae = NA,
      runtime = NA,
      success = FALSE
    ))
  })
}

# 主实验循环
n_experiments <- 100
missing_rates <- c(0.05, 0.1, 0.15)

all_results <- data.frame(
  Experiment = numeric(),
  Mechanism = character(),
  Missing_Rate = numeric(),
  MSE = numeric(),
  MAE = numeric(),
  Runtime = numeric(),
  Success = logical(),
  stringsAsFactors = FALSE
)

for(exp in 1:n_experiments) {
  cat(sprintf("\n执行实验 %d/%d\n", exp, n_experiments))
```

```r
  set.seed(exp * 123)  # 使用不同的种子

  # 加载并预处理数据
  data(mtcars)
  y <- mtcars$vs
  X <- scale(mtcars[, c("mpg", "disp")])

  # 创建完整数据的模型
  model_complete <- glm(y ~ X, family = binomial(link = "logit"))
  beta_complete <- coef(model_complete)

  for(mechanism in c("MCAR", "MAR")) {
    for(rate in missing_rates) {
      cat(sprintf("Running test: %s, Missing rate: %f\n", mechanism, rate))

      result <- run_test(X, y, mechanism, rate, beta_complete)

      all_results <- rbind(all_results, data.frame(
        Experiment = exp,
        Mechanism = mechanism,
        Missing_Rate = rate,
        MSE = result$mse,
        MAE = result$mae,
        Runtime = result$runtime,
        Success = result$success
      ))
    }
  }

  # 每10次实验保存一次结果
  if(exp %% 10 == 0) {
    write.csv(all_results,
              sprintf("simulation_results_checkpoint_%d.csv", exp),
              row.names = FALSE)
  }
}

# 分析成功的实验结果
successful_results <- subset(all_results, Success == TRUE)
```

```r
# 汇总统计
summary_stats <- successful_results %>%
  group_by(Mechanism, Missing_Rate) %>%
  summarise(
    n_success = n(),
    Mean_MSE = mean(MSE, na.rm = TRUE),
    SD_MSE = sd(MSE, na.rm = TRUE),
    Mean_MAE = mean(MAE, na.rm = TRUE),
    SD_MAE = sd(MAE, na.rm = TRUE),
    Mean_Runtime = mean(Runtime, na.rm = TRUE),
    Success_Rate = n() / n_experiments * 100,
    .groups = 'drop'
  )

print(summary_stats)
# 首先检查数据
str(successful_results)
summary(successful_results)

# 确保数据类型正确，并处理可能的NA值
successful_results <- successful_results %>%
  mutate(
    Missing_Rate = as.numeric(as.character(Missing_Rate)),
    MSE = as.numeric(as.character(MSE)),
    MAE = as.numeric(as.character(MAE))
  ) %>%
  filter(!is.na(MSE), !is.na(MAE))

# 重新创建MSE的箱线图，简化版本
p_mse <- ggplot(successful_results,
                aes(x = factor(Missing_Rate),
                    y = MSE,
                    fill = Mechanism)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "MSE Distribution",
       x = "Missing Rate",
       y = "MSE") +
```

```
  theme(legend.position = "top")
print(p_mse)
# 重新创建MAE的箱线图，简化版本
p_mae <- ggplot(successful_results,
                aes(x = factor(Missing_Rate),
                    y = MAE,
                    fill = Mechanism)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "MAE Distribution",
       x = "Missing Rate",
       y = "MAE") +
  theme(legend.position = "top")
print(p_mae)
```
不同方法比较
```{r}
library(mice)
library(Amelia)

# 生成模拟数据的函数
generate_data <- function(n, p, missing_rate = 0.2, seed = 123) {
  set.seed(seed)

  # 生成协变量
  X <- matrix(rnorm(n * p), nrow = n)

  # 真实参数
  beta_true <- c(1, rep(0.5, p))

  # 生成响应变量
  linear_pred <- cbind(1, X) %*% beta_true
  prob <- 1 / (1 + exp(-linear_pred))
  y <- rbinom(n, 1, prob)

  # 随机生成缺失值
  for(j in 1:p) {
    miss_ind <- sample(1:n, size = floor(n * missing_rate))
    X[miss_ind, j] <- NA
```

```
  }

  return(list(X = X, y = y, beta_true = beta_true))
}

# 评估函数
evaluate_method <- function(beta_est, beta_se, beta_true) {
  bias <- mean(beta_est - beta_true)
  rmse <- sqrt(mean((beta_est - beta_true)^2))
  coverage <- mean(beta_true >= (beta_est - 1.96*beta_se) &
                   beta_true <= (beta_est + 1.96*beta_se))

  return(c(bias = bias, rmse = rmse, coverage = coverage))
}

# 主模拟函数
run_simulation <- function(n_sim = 100, n = 200, p = 3, missing_rate = 0.2) {
  results <- list()

  # 存储结果的矩阵
  results_saem <- matrix(NA, n_sim, 3)
  results_mice <- matrix(NA, n_sim, 3)
  results_amelia <- matrix(NA, n_sim, 3)

  for(i in 1:n_sim) {
    # 生成数据
    data <- generate_data(n, p, missing_rate, seed = i)

    # 1. SAEM方法
    saem_fit <- miss.saem(data$X, data$y, pos_var = 1:p,
                          var_cal = TRUE, print_iter = FALSE)

    # 2. MICE方法
    mice_data <- data.frame(y = data$y)
    for(j in 1:p) {
      mice_data[paste0("X", j)] <- data$X[,j]
    }
    mice_imp <- mice(mice_data, m = 5, print = FALSE)
```

```
  # 创建公式
  formula_str <- paste("y ~", paste(paste0("X", 1:p), collapse = " + "))
  mice_fits <- with(mice_imp, glm(as.formula(formula_str), family = binomial()))
  mice_pooled <- pool(mice_fits)

  # 3. Amelia方法
  amelia_data <- mice_data
  amelia_imp <- amelia(amelia_data, m = 5, p2s = 0)
  amelia_results <- matrix(NA, 5, p + 1)

  for(j in 1:5) {
    imp_data <- amelia_imp$imputations[[j]]
    fit <- glm(as.formula(formula_str), data = imp_data, family = binomial())
    amelia_results[j,] <- coef(fit)
  }

  # 评估结果
  results_saem[i,] <- evaluate_method(saem_fit$beta,
                                      saem_fit$std_obs,
                                      data$beta_true)

  results_mice[i,] <- evaluate_method(summary(mice_pooled)$estimate,
                                      summary(mice_pooled)$std.error,
                                      data$beta_true)

  results_amelia[i,] <- evaluate_method(colMeans(amelia_results),
                                        apply(amelia_results, 2, sd),
                                        data$beta_true)

  if(i %% 10 == 0) cat("Completed simulation", i, "of", n_sim, "\n")
}

# 整理结果
results$saem <- colMeans(results_saem)
results$mice <- colMeans(results_mice)
results$amelia <- colMeans(results_amelia)

names(results$saem) <- names(results$mice) <- names(results$amelia) <-
  c("Bias", "RMSE", "Coverage")
```

```r
  return(results)
}

# 运行模拟
set.seed(123)
sim_results <- run_simulation(n_sim = 100, n = 200, p = 3, missing_rate = 0.2)

# 打印结果
print("SAEM Results:")
print(round(sim_results$saem, 4))
print("MICE Results:")
print(round(sim_results$mice, 4))
print("Amelia Results:")
print(round(sim_results$amelia, 4))

# 可视化结果
library(ggplot2)
library(reshape2)

results_df <- data.frame(
  Method = rep(c("SAEM", "MICE", "Amelia"), each = 3),
  Metric = rep(c("Bias", "RMSE", "Coverage"), 3),
  Value = c(sim_results$saem, sim_results$mice, sim_results$amelia)
)

ggplot(results_df, aes(x = Method, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~Metric, scales = "free") +
  theme_minimal() +
  labs(title = "Comparison of Methods",
       y = "Value",
       x = "Method")
```