

## PCG for the infinite-dimensional mode- $k$ subproblem (missing data)

We consider the linear system in the unknown  $W \in \mathbb{R}^{n \times r}$

$$[(Z \otimes K)^\top SS^\top(Z \otimes K) + \lambda(I_r \otimes K)] \text{vec}(W) = (I_r \otimes K) \text{vec}(B), \quad (1)$$

where  $K \in \mathbb{R}^{n \times n}$  is a (symmetric) psd kernel matrix,  $Z \in \mathbb{R}^{M \times r}$  is the Khatri–Rao product of the other factors,  $S \in \mathbb{R}^{N \times q}$  selects the  $q$  observed entries (so  $S^\top \text{vec}(T)$  equals the observed values), and  $B = TZ \in \mathbb{R}^{n \times r}$ . Throughout we assume  $n, r < q \ll N = nM$  and avoid any  $O(N)$  work.

**1. Variational form, symmetry, and positive definiteness.** Let  $P \equiv SS^\top \in \mathbb{R}^{N \times N}$  be the diagonal “mask” matrix that keeps observed entries and zeros missing ones (so  $P = P^\top = P^2$ ). The system (1) is the normal equation for the regularized least-squares objective

$$\min_{W \in \mathbb{R}^{n \times r}} \frac{1}{2} \|S^\top \text{vec}(T) - S^\top \text{vec}(KWZ^\top)\|_2^2 + \frac{\lambda}{2} \text{Tr}(W^\top KW) = \frac{1}{2} \|P \circ (T - KWZ^\top)\|_F^2 + \frac{\lambda}{2} \text{Tr}(W^\top KW), \quad (2)$$

where  $\circ$  denotes Hadamard product and we used  $\text{Tr}(W^\top KW) = \|W\|_{\mathcal{H}}^2$  as the RKHS penalty. Writing the first term as  $\frac{1}{2} \|P^{1/2}(\text{vec}(T) - (Z \otimes K) \text{vec}(W))\|_2^2$  shows the Hessian is

$$A \equiv (Z \otimes K)^\top P(Z \otimes K) + \lambda(I_r \otimes K) \in \mathbb{R}^{nr \times nr}, \quad b \equiv (I_r \otimes K) \text{vec}(B) = \text{vec}(KB).$$

Thus  $A$  is symmetric. If  $K \succ 0$  and  $\lambda > 0$ , then  $A \succ 0$  because for any  $x \neq 0$ ,

$$x^\top Ax = \|P^{1/2}(Z \otimes K)x\|_2^2 + \lambda x^\top(I_r \otimes K)x \geq \lambda x^\top(I_r \otimes K)x > 0.$$

(If  $K$  is only psd, one can add a nugget  $\varepsilon I_n$  to  $K$  or instead regularize with  $\lambda(I_r \otimes I_n)$ ; we proceed with the  $K \succ 0$  case.) Hence we can solve (1) with (preconditioned) conjugate gradients (CG/PCG).

**2. Why PCG helps.** Direct solution would require forming  $A$  and performing a dense factorization costing  $O((nr)^3) = O(n^3r^3)$ . In contrast, PCG requires only: (i) repeated matrix–vector products  $y \leftarrow Ax$  and (ii) repeated applications of a preconditioner  $M^{-1}$ , with overall cost  $\approx \#\text{iters} \times (\text{matvec} + \text{precond})$ . Our goal is to implement both in  $O(n^2r + qr)$  time per iteration and memory  $O(nr + qr)$ , never touching  $N$ -scale arrays.

**3. PCG (brief).** Choose an SPD preconditioner  $M \approx A$  that is cheap to invert. Starting from  $x_0$  (often 0 or the previous ALS iterate), define  $r_0 = b - Ax_0$  and solve  $Mz_0 = r_0$ . Set  $p_0 = z_0$  and for  $t = 0, 1, 2, \dots$  iterate

$$\alpha_t = \frac{\langle r_t, z_t \rangle}{\langle p_t, Ap_t \rangle}, \quad x_{t+1} = x_t + \alpha_t p_t, \quad r_{t+1} = r_t - \alpha_t Ap_t, \quad \text{solve } Mz_{t+1} = r_{t+1}, \quad \beta_t = \frac{\langle r_{t+1}, z_{t+1} \rangle}{\langle r_t, z_t \rangle}, \quad p_{t+1} = z_{t+1} + \beta_t p_t$$

The algorithm only needs the ability to compute  $Ap_t$  (matvec) and to apply  $M^{-1}$ .

## Efficient matrix–vector products without forming $A$

**Operator viewpoint.** Define the linear map  $\mathcal{L} : \mathbb{R}^{n \times r} \rightarrow \mathbb{R}^q$  by

$$(\mathcal{L}(X))_t \equiv (KXZ^\top)_{i_t, j_t}, \quad t = 1, \dots, q.$$

Then the data term in (2) is  $\frac{1}{2}\|\mathcal{L}(W) - y\|_2^2$  with  $y \equiv S^\top \text{vec}(T)$ , and the normal equation is

$$(\mathcal{L}^\top \mathcal{L} + \lambda \mathcal{R}) \text{vec}(W) = \mathcal{L}^\top y, \quad \mathcal{R} = I_r \otimes K.$$

Crucially, both  $\mathcal{L}(X)$  (gather) and  $\mathcal{L}^\top u$  (scatter + multiply by  $Z$  and  $K$ ) can be applied in  $O(qr + n^2r)$  time.

Represent an input vector  $x \in \mathbb{R}^{nr}$  as a matrix  $X \in \mathbb{R}^{n \times r}$  such that  $x = \text{vec}(X)$ . Use the identity

$$(Z \otimes K) \text{vec}(X) = \text{vec}(K X Z^\top), \quad (3)$$

which is a special case of  $\text{vec}(AXB^\top) = (B \otimes A) \text{vec}(X)$ .

### Observed-entry operator implemented with index lists

Let the observed entries of the mode- $k$  unfolding be indexed by pairs  $(i_t, j_t)$  for  $t = 1, \dots, q$  with  $i_t \in [n]$  and  $j_t \in [M]$ . Then for any matrix  $U \in \mathbb{R}^{n \times M}$ ,

$$S^\top \text{vec}(U) = (U_{i_t, j_t})_{t=1}^q \in \mathbb{R}^q, \quad \text{and} \quad \text{reshape}_{n \times M}(Sv) \text{ has nonzeros } (i_t, j_t) \text{ equal to } v_t.$$

Thus we can realize  $S^\top$  (gather) and  $S$  (scatter) in  $O(q)$  time using stored index arrays  $(i_t, j_t)$ .

**Avoiding explicit formation of  $Z$ .** Although  $Z \in \mathbb{R}^{M \times r}$  is defined as a Khatri–Rao product,  $M = \prod_{i \neq k} n_i$  can be enormous, so we do *not* store  $Z$ . Instead, for each observed tensor entry we typically store its full multi-index  $(i_1^{(t)}, \dots, i_d^{(t)})$ ; the corresponding row needed in (4) and (7) is

$$Z_{j_t,:} = A_d(i_d^{(t)}, :) \odot \dots \odot A_{k+1}(i_{k+1}^{(t)}, :) \odot A_{k-1}(i_{k-1}^{(t)}, :) \odot \dots \odot A_1(i_1^{(t)}, :),$$

which can be computed on the fly in  $O((d-1)r)$  time per observed entry (or faster if intermediate Hadamard products are cached). This keeps both memory and time independent of  $M$ .

### Matvec formula

Given  $X \in \mathbb{R}^{n \times r}$ , compute  $Y \in \mathbb{R}^{n \times r}$  so that  $\text{vec}(Y) = A \text{vec}(X)$ . Write  $G \equiv KX \in \mathbb{R}^{n \times r}$ . Then the observed predicted entries of  $U \equiv KXZ^\top \in \mathbb{R}^{n \times M}$  are

$$u_t \equiv U_{i_t, j_t} = G_{i_t,:} \cdot Z_{j_t,:} \quad (t = 1, \dots, q), \quad (4)$$

each a length- $r$  dot product. Now form the sparse matrix  $\tilde{U} \in \mathbb{R}^{n \times M}$  with  $(\tilde{U})_{i_t, j_t} = u_t$  and all other entries zero (this is exactly  $\text{reshape}(SS^\top \text{vec}(U))$ ). Finally apply  $(Z \otimes K)^\top$  using the transpose identity

$$(Z \otimes K)^\top \text{vec}(\tilde{U}) = \text{vec}(K \tilde{U} Z), \quad (5)$$

obtaining the main term  $K(\tilde{U}Z)$ . Adding the Tikhonov term gives

$$Y = K(\tilde{U}Z) + \lambda KX. \quad (6)$$

**Proof of (6).** Let  $x = \text{vec}(X)$ . The first term satisfies  $(Z \otimes K)^\top P(Z \otimes K)x = (Z \otimes K)^\top \text{vec}(\tilde{U})$  with  $\tilde{U} \equiv \text{reshape}_{n \times M}(P \text{vec}(KXZ^\top))$ . Using (5) yields  $\text{vec}(K\tilde{U}Z)$ , i.e., the matrix  $K(\tilde{U}Z)$ . The regularizer contributes  $\lambda(I_r \otimes K) \text{vec}(X) = \lambda \text{vec}(KX)$ , giving (6).

## How to compute $\tilde{U}Z$ in $O(qr)$

We never form  $\tilde{U}$  explicitly as an  $n \times M$  array. Instead, compute the product  $H \equiv \tilde{U}Z \in \mathbb{R}^{n \times r}$  by accumulating contributions from the  $q$  nonzeros:

$$H_{i_t,:} += u_t Z_{j_t,:} \quad (t = 1, \dots, q). \quad (7)$$

Each update is a SAXPY of length  $r$ , so the cost is  $O(qr)$ . Then compute  $KH$  in  $O(n^2r)$  time (dense  $K$ ), and add  $\lambda KX$ .

**Implementation sketch (matvec).** Given  $X \in \mathbb{R}^{n \times r}$ , precompute  $G \leftarrow KX$ . Initialize  $H \leftarrow 0 \in \mathbb{R}^{n \times r}$ . Loop over observed entries  $t = 1, \dots, q$ : compute (or fetch) the Khatri–Rao row  $z_t \equiv Z_{j_t,:}$ , compute the scalar  $u_t \leftarrow G_{i_t,:} z_t^\top$ , and update  $H_{i_t,:} \leftarrow H_{i_t,:} + u_t z_t$ . Finally return  $Y \leftarrow KH + \lambda G$ .

**Matvec complexity.** Assuming dense  $K$ :

- $G = KX$ :  $O(n^2r)$ .
- gather  $u_t$  via (4):  $O(qr)$  given access to  $Z_{j_t,:}$ ; if  $Z_{j_t,:}$  is computed on the fly from the CP factors, add  $O(q(d-1)r)$ .
- accumulate  $H = \tilde{U}Z$  via (7):  $O(qr)$  (plus the same cost to form  $Z_{j_t,:}$  if needed).
- $KH$  and  $\lambda KX$ :  $O(n^2r)$  (can reuse  $G$  for  $KX$ ).

Total per matvec:  $O(n^2r + qr)$  time if  $Z$ -rows are available (or  $O(n^2r + qdr)$  if computed on the fly),  $O(nr + qr)$  memory for  $X, G, H$  and index lists; crucially independent of  $N = nM$ .

## Computing the right-hand side $b$ without forming $T$

The right-hand side is  $b = \text{vec}(KB)$ . We can compute  $B = TZ$  using only observed entries: if the observed tensor value at  $(i_t, j_t)$  is  $t_t$ , then

$$B_{i_t,:} += t_t Z_{j_t,:},$$

which costs  $O(qr)$  given access to  $Z_{j_t,:}$  (or  $O(qdr)$  if each  $Z_{j_t,:}$  is formed on the fly from the CP factors), followed by  $KB$  in  $O(n^2r)$ . (This is the same sparse accumulation pattern as (7).)

## Preconditioning

### Kronecker “full-observation” preconditioner

A standard and effective choice is to drop the mask  $P$  (equivalently, pretend all entries are observed). Then

$$A_0 \equiv (Z \otimes K)^\top (Z \otimes K) + \lambda(I_r \otimes K) = (Z^\top Z) \otimes (K^\top K) + \lambda(I_r \otimes K) = (Z^\top Z) \otimes (K^2) + \lambda(I_r \otimes K),$$

where  $K^2$  denotes the usual matrix product  $KK$  (since  $K$  is symmetric). Let  $G \equiv Z^\top Z \in \mathbb{R}^{r \times r}$ . Since  $Z$  is a Khatri–Rao product,  $G$  can be computed without forming  $Z$  via the Hadamard product (denoted  $*$ ) of Gram matrices,

$$G = \underset{i \neq k}{*} (A_i^\top A_i),$$

costing  $O(\sum_{i \neq k} n_i r^2)$ . If we precompute eigendecompositions

$$K = U\Lambda U^\top, \quad G = V\Sigma V^\top,$$

then  $A_0$  diagonalizes in the Kronecker basis:

$$A_0 = (V \otimes U) \operatorname{diag}(\sigma_a \lambda_b^2 + \lambda \lambda_b)_{a \in [r], b \in [n]} (V \otimes U)^\top.$$

Hence applying  $M^{-1} \approx A_0^{-1}$  to a vector  $x = \operatorname{vec}(X)$  can be done by:

1. transform  $\hat{X} \leftarrow U^\top X V$  (two small dense multiplies),
2. elementwise divide  $\hat{X}_{b,a} \leftarrow \hat{X}_{b,a} / (\sigma_a \lambda_b^2 + \lambda \lambda_b)$ ,
3. inverse transform  $X \leftarrow U \hat{X} V^\top$ .

This costs  $O(n^2 r + nr^2)$  per application (often dominated by  $O(n^2 r)$  when  $n \geq r$ ), after one-time setup  $O(n^3 + r^3)$ .

**Why this is reasonable.**  $A$  equals  $A_0$  when  $P = I_N$ ; when entries are missing uniformly at random,  $\mathbb{E}[P] = (q/N)I_N$ , so  $\mathbb{E}[(Z \otimes K)^\top P (Z \otimes K)] = (q/N)(Z^\top Z \otimes K^2)$  is a scaled version of the same Kronecker term. Thus  $A_0$  (or the scaled variant  $\alpha A_0$  with  $\alpha = q/N$ ) captures the dominant spectral structure, while  $P$  acts as a perturbation; PCG corrects the mismatch through iterations.

## Simpler (cheaper) preconditioners

If eigendecompositions are too costly, a cheaper alternative is a block-diagonal preconditioner

$$M_{\text{bd}} = (\operatorname{diag}(G) \otimes K^2) + \lambda(I_r \otimes K),$$

which decouples the  $r$  components, requiring  $r$  solves with  $n \times n$  matrices of the form  $(g_{\ell\ell} K^2 + \lambda K) = K(g_{\ell\ell} K + \lambda I)$ . If  $K$  is factored once (Cholesky), these are fast; if  $K$  is large, one can use an approximate factorization (pivoted Cholesky / incomplete Cholesky) as a preconditioner for these inner solves.

## Overall complexity and scaling

Let  $m$  be the number of PCG iterations to reach a desired tolerance; standard theory gives  $m = O(\sqrt{\kappa(M^{-1}A)} \log(1/\varepsilon))$  for relative error  $\varepsilon$ , so a good preconditioner aims to make  $\kappa(M^{-1}A)$  close to 1.

- One-time setup: compute  $Z$  and (optionally)  $G = Z^\top Z$  in  $O(Mr^2)$  if done naively, but in CP-ALS contexts  $Z$  is implicit and  $G$  is usually assembled from Hadamard products of Gram matrices of each factor at cost  $O(\sum_{i \neq k} n_i r^2)$ , avoiding  $M$ .
- Right-hand side: compute  $B$  in  $O(qr)$  and then  $KB$  in  $O(n^2 r)$ .
- Each PCG iteration:
  - matvec  $Ax$ :  $O(n^2 r + qr)$ .
  - preconditioner apply (Kronecker-eig):  $O(n^2 r + nr^2)$ .
  - vector updates/inner products:  $O(nr)$ .

Hence total time  $O(m(n^2 r + qr + nr^2) + qr + n^2 r)$ , which is dramatically better than  $O(n^3 r^3)$  when  $m \ll n^2 r^2$  and  $q \ll N$ .

**Key point: no  $N$ -scale work.** All operations are expressed in terms of  $n, r, q$  and small Gram matrices; selection/scatter uses only the  $q$  observed indices and values.

**Remark (faster kernel multiplies).** If  $K$  admits a fast matrix–vector/matrix multiply (e.g., via Nyström, inducing points, random features, or a structured kernel), then the  $O(n^2r)$  terms above can be reduced accordingly; the  $O(qr)$  sparse gather/accumulate terms remain unchanged.