Computing the Multivariate Hypergeometric Function

Plamen Koev
Department of Mathematics
M.I.T.

Numerical Linear Algebra Seminar, M.I.T., April 9, 2004

The Hypergeometric Function

- **Univariate**

$$ {}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; x) \equiv \sum_{k=0}^{\infty} \frac{(a_1)_k \ldots (a_p)_k}{k!(b_1)_k \ldots (b_q)_k} \cdot x^k, $$

  where $(a)_k = a(a+1)\ldots(a+k-1)$.

- **Hard problem to approximate. Slow convergence**

The Hypergeometric Function II

- **Univariate**

$$ {}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;x) \equiv \sum_{k=0}^{\infty} \frac{(a_1)_k \ldots (a_p)_k}{k!(b_1)_k \ldots (b_q)_k} \cdot x^k, $$

  where $(a)_k = a(a+1)\ldots(a+k-1)$.

- **Hard problem to approximate. Slow convergence**

- **Multivariate**

$$ {}_pF_q^{\alpha}(a_1,\ldots,a_p;b_1,\ldots,b_q;x_1,\ldots,x_n) $$
$$ \equiv \sum_{k=0}^{\infty} \sum_{\kappa \vdash k} \frac{(a_1)_\kappa \ldots (a_p)_\kappa}{k!(b_1)_\kappa \ldots (b_q)_\kappa} \cdot C_\kappa^{\alpha}(x_1,\ldots,x_n), $$

  where

  - $(a)_\kappa \equiv \prod_{(i,j)\in\kappa} \left(a - \frac{i-1}{\alpha} + j - 1\right)$ —Pocchammer symbol
  - $C_\kappa^{\alpha}(x_1, x_2, \ldots, x_n)$ is the "C" Jack Function

An Application—p.d.f. of $\lambda_{\max}$ of a β-Laguerre matrix

Definition: $L = BB^T$, where

$$B = \begin{bmatrix} \chi_{2a} & & & \\ \chi_{\beta(n-1)} & \chi_{2a-\beta} & & \\ & \ddots & \ddots & \\ & & \chi_{\beta} & \chi_{2a-\beta(n-1)} \end{bmatrix},$$

and $k = a - \frac{\beta}{2}(n-1) - 1$ is a nonnegative integer.

The p.d.f. of $\lambda_{\max}$ is

$$f(x) = x^{kn} \cdot e^{-\frac{nx}{2}} \cdot \ {}_2F_0^{2/\beta}(-k, \beta\frac{n}{2} + 1; ; -\frac{2}{x} I_{n-1}).$$

Partitions

$$\kappa = (\kappa_1, \kappa_2, \ldots) \vdash k$$

Means

$$k = \kappa_1 + \kappa_2 + \ldots \quad \text{and} \quad \kappa_1 \geq \kappa_2 \geq \ldots$$

Example:

$$\begin{aligned}
1 &= 1 \\
2 &= 2 \\
&= 1+1 \\
3 &= 3 \\
&= 2+1 \\
&= 1+1+1 \\
4 &= 4 \\
&= 3+1 \\
&= 2+2 \\
&= 2+1+1 \\
&= 1+1+1+1
\end{aligned}$$

Complexity of Computing a Truncation

- **Univariate**

$$ {}_pF_q(a_{1:p}; b_{1:q}; x) \approx \sum_{k=0}^{m} \frac{(a_1)_k \dots (a_p)_k}{k!(b_1)_k \dots (b_q)_k} \cdot x^k $$

  — m terms.

Complexity of Computing a Truncation

- Univariate

$$ {}_pF_q(a_{1:p}; b_{1:q}; x) \approx \sum_{k=0}^{m} \frac{(a_1)_k \dots (a_p)_k}{k!(b_1)_k \dots (b_q)_k} \cdot x^k $$

— m terms.

- Multivariate

$$ {}_p F_q^{\alpha}(a_{1:p}; b_{1:q}; x_{1:n}) \approx \sum_{k=0}^{m} \sum_{\kappa \vdash k} \frac{(a_1)_\kappa \dots (a_p)_\kappa}{k!(b_1)_\kappa \dots (b_q)_\kappa} \cdot C_\kappa^{\alpha}(x_{1:n}), $$

$\sim e^{\sqrt{m}} n^m$ terms.

Because $\#\{|\kappa| \le m\} \approx e^{\sqrt{m}}$ and $C_\kappa^{\alpha}(x_{1:n})$ has $\sim n^{|\kappa|}$ terms.

Complexity of Computing a Truncation

- Univariate

$$ {}_pF_q(a_{1:p}; b_{1:q}; x) \approx \sum_{k=0}^{m} \frac{(a_1)_k \dots (a_p)_k}{k!(b_1)_k \dots (b_q)_k} \cdot x^k $$

— m terms.

- Multivariate

$$ {}_p F_q^{\alpha}(a_{1:p}; b_{1:q}; x_{1:n}) \approx \sum_{k=0}^{m} \sum_{\kappa \vdash k} \frac{(a_1)_\kappa \dots (a_p)_\kappa}{k!(b_1)_\kappa \dots (b_q)_\kappa} \cdot C_\kappa^{\alpha}(x_{1:n}), $$

$\sim e^{\sqrt{m}} n^m$ terms.

Because $\#\{|\kappa| \le m\} \approx e^{\sqrt{m}}$ and $C_\kappa^{\alpha}(x_{1:n})$ has $\sim n^{|\kappa|}$ terms.

- In contrast, new algorithm cost bounded by

$$ O(e^m \cdot n) $$

Idea: Update, do not Compute

$$ {}_p F_q^{\alpha}(a_{1:p}; b_{1:q}; x_{1:n}) \approx \sum_{k=0}^{m} \sum_{\kappa \vdash k} \underbrace{\frac{(a_1)_\kappa \dots (a_p)_\kappa}{k!(b_1)_\kappa \dots (b_q)_\kappa}}_{Q_\kappa} \cdot C_\kappa^{\alpha}(x_{1:n}) $$

Store every computed Q_κ and C_κ^{α}.

If

$$ \begin{aligned} \kappa &= (\kappa_1, \kappa_2, \dots, \kappa_i, \dots) \\ \nu &= (\kappa_1, \kappa_2, \dots, \kappa_i - 1, \dots) \end{aligned} $$

Then

$$ \frac{Q_\kappa}{Q_\nu} = \frac{\prod_{j=1}^{p} (a_j + c)}{\prod_{j=1}^{q} (b_j + c)}, \quad \text{where} \quad c = -\frac{i-1}{\alpha} + \kappa_i - 1; $$

Cost: $2(p+q)$ instead of $|\kappa|(p+q)$.

Idea: Update, do not Compute

For the Jack function, look at the Schur function first ($\alpha = 1$).

$$\begin{aligned} s_{(1)}(x_{1:n}) &= x_1 + \ldots + x_n \\ s_{(2)}(x_{1:n}) &= \sum_{i \le j} x_i x_j \\ s_{(2,1)}(x_{1:n}) &= \sum_{i \neq j} x_i^2 x_j + 2 \sum_{i<j<k} x_i x_j x_k \\ s_{(1,1,1)}(x_{1:n}) &= \sum_{i<j<k} x_i x_j x_k \end{aligned}$$

In general:

$$\begin{aligned} s_\kappa(x_{1:n}) &= \sum_T x^T \\ C_\kappa^\alpha(x_{1:n}) &= \sum_T f_T(\alpha) x^T, \end{aligned}$$

where

- T ranges over all SSYT of shape κ
- $f_T(\alpha)$ is a rational function of α

Idea in computing s_κ efficiently

$$
\begin{aligned}
s_{(2)} &= \sum_{i \le j} x_i x_j \\
&= x_1 \cdot x_1 + x_2(x_1 + x_2) + x_3(x_1 + x_2 + x_3) + \ldots + x_n(x_1 + \ldots + x_n)
\end{aligned}
$$

Algorithm:

1. Precompute $s_i = \Sigma_{j=1}^{i} x_i$
2. $s_{(2)} = x_1 s_1 + x_2 s_2 + \ldots x_n s_n$

Cost $O(n)$ instead of $O(n^2)$.

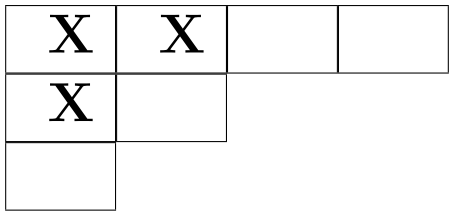
Idea: Update, do not Compute

$$ {}_p F_q^{\alpha}(a_{1:p}; b_{1:q}; x_{1:n}) \approx \sum_{k=0}^{m} \sum_{\kappa \vdash k} \underbrace{\frac{(a_1)_\kappa \dots (a_p)_\kappa}{k!(b_1)_\kappa \dots (b_q)_\kappa}}_{Q_\kappa} \cdot C_\kappa^{\alpha}(x_{1:n}) $$

$$ C_\kappa^{\alpha}(x_{1:n}) = \sum_{\mu \leq \kappa} C_\mu^{\alpha}(x_{1:n-1}) \cdot x_n^{|\kappa|-|\mu|} \cdot \gamma_{\kappa\mu}, $$

summation over κ/μ–horizontal strip, meaning

$$ \mu_1 \geq \kappa_2; \quad \mu_2 \geq \kappa_3; \quad \mu_3 \geq \kappa_4 \dots $$



Fewer than $M = \#\{|\kappa| \leq m\}$ terms above. Need to compute for all $|\kappa| \leq m$ and $1, 2, \dots, n$, so complexity overall bounded by

$$ M^2 \cdot n \approx e^m \cdot n $$

How do we store C_{κ}^{α} in memory efficiently?

- **How do we index $|\kappa| \leq m$?**

N_κ	κ	$D(N_\kappa)$
1	(1)	5
2	(2)	6
3	(3)	8
4	(4)	
5	(1, 1)	9
6	(2, 1)	10
7	(2, 2)	
8	(3, 1)	
9	(1, 1, 1)	11
10	(2, 1, 1)	
11	(1, 1, 1, 1)	

$$N_{(\kappa_1)} = \kappa_1 + 1$$
$$N_{(\kappa_1,\ldots,\kappa_i)} = D(N_{(\kappa_1,\ldots,\kappa_{i-1})}) + \kappa_i - 1, \text{ when } i > 1 \text{ and } \kappa_i > 0.$$

Easy case: $x_1 = x_2 = \ldots = x_n$

Then (in "J" normalization)

$$J^{\alpha}_{\kappa}(xI_n) = x^{|\kappa|} \prod_{(i,j)\in\kappa} (n - (i-1) + \alpha(j-1)),$$

which implies $J^{\alpha}_{\kappa}(xI_n) = J^{\alpha}_{\nu}(xI_n) \cdot x \cdot (n - (i-1) + \alpha(\kappa_i - 1))$.

No need to store C^{α}_{κ} in this case.

```
function s =hgi(m, n, x, α, a, b)
  s = 1
  summation(1, 0, 1, m);
function summation(i, κ, z, j)
  for κ_i = 1 : min(κ_{i−1}, j)  ... defaults to j for i = 1
    c = −(i − 1)/α + j − 1
    z = (z. ∗ x)(n + αc)prod(a + c)/prod(b + c) · j_κ/j_ν
    s = s + z
    if j > κ_i then summation (i + 1, κ, z, j − κ_i)
  endfor
```