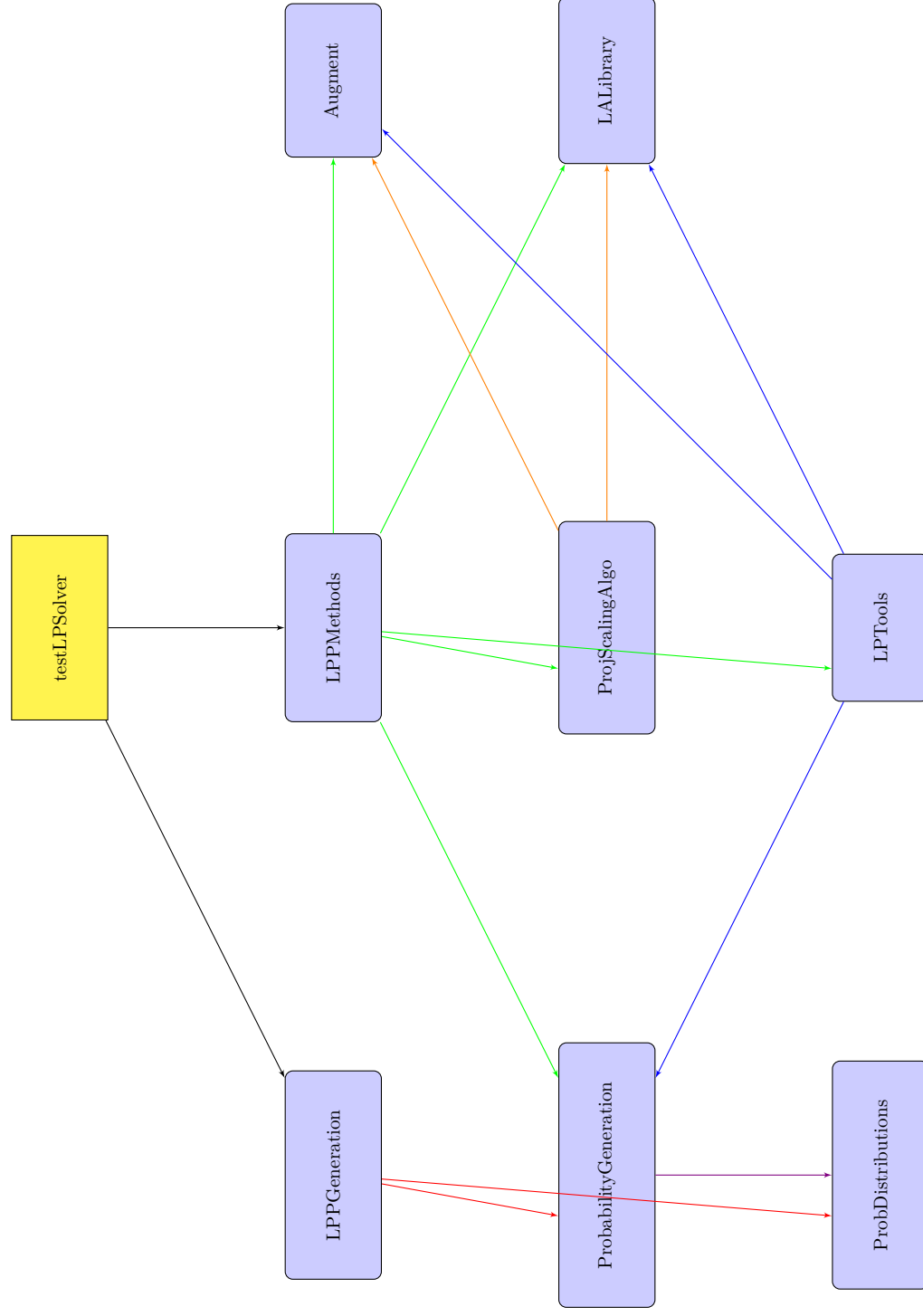# LP Solver
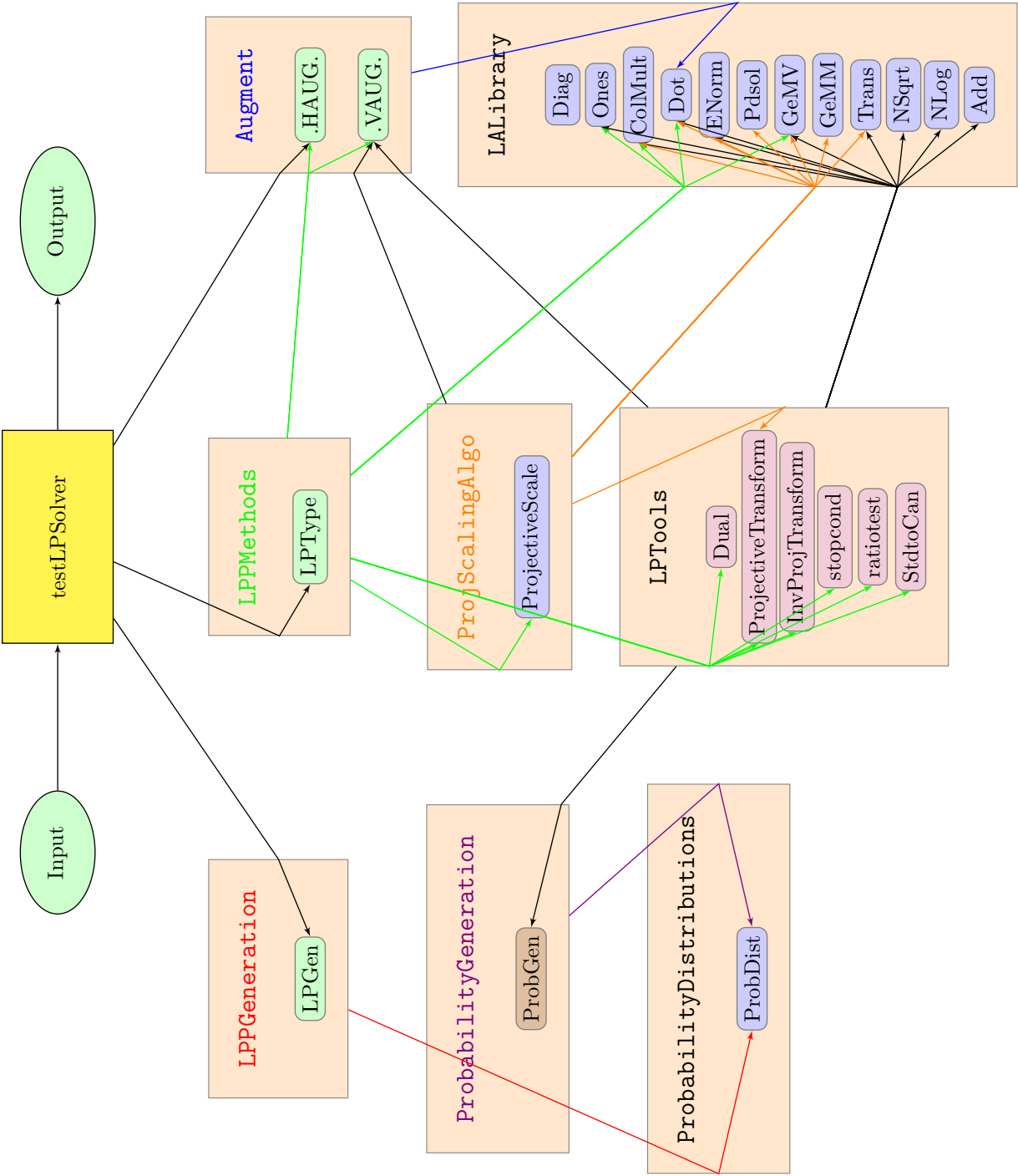
Akhil Akkapelli

January 6, 2022
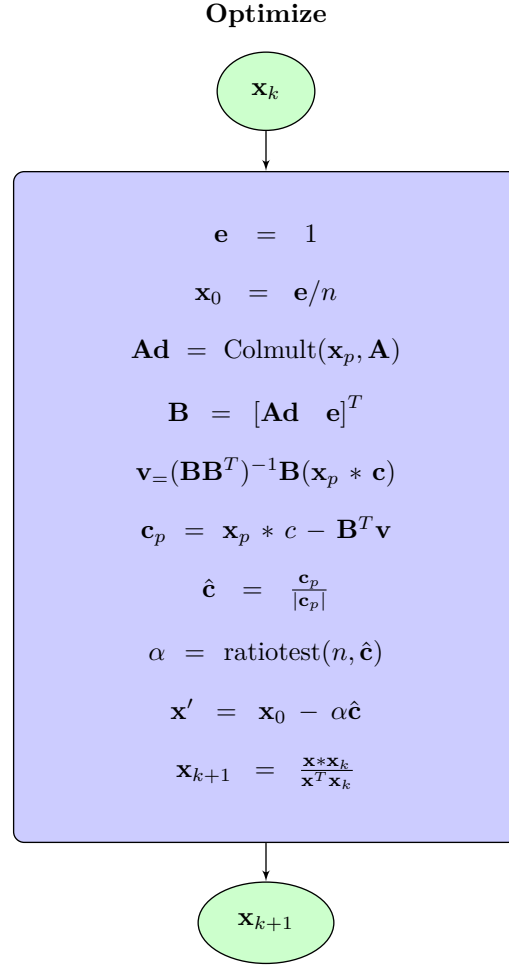
0.1 Simple Module Dependency Diagram

0.2 Detailed Modular Dependency Diagram

1 Projective Scaling Algorithm

Optimize

$$\mathbf{x}_k$$

$$
\begin{aligned}
\mathbf{e} &= 1 \\
\mathbf{x}_0 &= \mathbf{e}/n \\
\mathbf{Ad} &= \mathrm{Colmult}(\mathbf{x}_p, \mathbf{A}) \\
\mathbf{B} &= [\mathbf{Ad} \quad \mathbf{e}]^T \\
\mathbf{v} &= (\mathbf{BB}^T)^{-1}\mathbf{B}(\mathbf{x}_p * \mathbf{c}) \\
\mathbf{c}_p &= \mathbf{x}_p * c - \mathbf{B}^T\mathbf{v} \\
\hat{\mathbf{c}} &= \frac{\mathbf{c}_p}{|\mathbf{c}_p|} \\
\alpha &= \mathrm{ratiotest}(n, \hat{\mathbf{c}}) \\
\mathbf{x}' &= \mathbf{x}_0 - \alpha\hat{\mathbf{c}} \\
\mathbf{x}_{k+1} &= \frac{\mathbf{x}*\mathbf{x}_k}{\mathbf{x}^T\mathbf{x}_k}
\end{aligned}
$$

$$\mathbf{x}_{k+1}$$

```fortran
1  PURE FUNCTION ProjectiveScale(A,c, stopcond, ratiotest) RESULT(xcan)
2  ! Obtain vector xcan by minimizing along Objective vector c of system Axcan = 0 using
      stoping condition stopcond and ratio test ratiotest functions
3
4  UREAL, INTENT(IN) :: A(:,:), c(:)
5  UREAL            :: xcan(size(A,2))
6
7  UREAL :: xp(size(A,2)), x(size(A,2))
8  INTEGER :: n, iter
9
10 INTERFACE
11
12    UREAL PURE FUNCTION ratiotest(n, cunit) RESULT(alpha)
13      INTEGER, INTENT(IN) :: n
14      UREAL,   INTENT(IN) :: cunit(n)
15      END FUNCTION ratiotest
16
17      LOGICAL PURE FUNCTION stopcond(n, x,xp, iter, c) RESULT(stp)
18    INTEGER, INTENT(IN) :: iter, n
19    UREAL, INTENT(IN) :: x(n), xp(n), c(n)
20    END FUNCTION stopcond
21
22 END INTERFACE
23
24
```

```fortran
25  !IF(size(A,2) /= size(c)) STOP "Algorithm ERROR: Wrong size for input argument"
26
27  n = size(A,2)
28  xp = one/n
29
30  !IF( ANY(GEMV(A,xp) >= ukind(1.Q-10)) ) STOP "Center of Simplex doesn't lie in Null space
        of the Input Matrix:"
31
32  iter = 1
33  DO
34
35  x = Optimize(xp)
36  IF(stopcond(n, x,xp, iter, c)) EXIT
37
38  !IF( ANY(ISNAN(x))) THEN
39  !PRINT*,  "WARNING: NAN occurred in the Solution"
40  !x= xp
41  !EXIT
42  !END IF
43
44  xp = x
45  iter = iter + 1
46
47  END DO
48
49  xcan = x
50
51  CONTAINS
52
53  PURE FUNCTION Optimize(xp) RESULT(x)
54
55  UREAL, INTENT(IN) :: xp(:)
56  UREAL            :: x(size(xp))
57
58  UREAL :: Ad(size(A,1),size(xp)), B(size(A,1)+1,size(xp)), &
59             v(size(A,1)+1), cp(size(c)), cunit(size(c)), alpha
60  UREAL ::  e(size(xp)) = one
61  UREAL :: x0(size(xp)) = e/n
62
63
64  Ad = COLMULT(xp,A)
65  B = Ad .VAUG. e
66  v = PDSOL(GEMM(B,TRANS(B)), GEMV(B,xp*c))
67
68  cp = xp*c - GEMV(TRANS(B),v)
69  cunit = cp/ENORM(cp)
70
71  alpha = ratiotest(n, cunit)
72
73  x = x0 - alpha*cunit
74  x = xp*x/DOT(xp,x)
75
76  END FUNCTION Optimize
77
78  END FUNCTION ProjectiveScale
```

ProjScalingAlgo

4

2 Ratio Test

```fortran
1 UREAL PURE FUNCTION potentialratio(n, cunit) RESULT(alpha)
2
3 INTEGER, INTENT(IN) :: n
4 UREAL, INTENT(IN) :: cunit(n)
5
6
7 alpha = 1/(4*NSQRT(ukind(n)*(ukind(n)-1)))
8
9 END FUNCTION potentialratio
```

PotentialRatio

```fortran
1 UREAL PURE FUNCTION minratio(n, cunit) RESULT(alpha)
2
3 INTEGER, INTENT(IN) :: n
4 UREAL, INTENT(IN) :: cunit(n)
5
6 UREAL, PARAMETER :: beta = ukind(1.Q-1)
7
8
9 alpha = (one-beta)/(n*maxval(cunit))
10
11 END FUNCTION minratio
```

MinimumRatio

```fortran
1 UREAL PURE FUNCTION zeroratio(n, cunit) RESULT(alpha)
2
3 INTEGER, INTENT(IN) :: n
4 UREAL, INTENT(IN) :: cunit(n)
5
6 UREAL, PARAMETER :: beta = ukind(1.Q-1)
7 UREAL :: a
8 INTEGER :: idx
9
10
11 alpha = one/(n*cunit(n-1))
12 DO idx=1,n
13   IF(cunit(idx)<0 .OR. idx == n-1) CYCLE
14   a = (ukind(1.Q0)-beta)/(n*cunit(idx))
15   IF(alpha>a) alpha = a
16 END DO
17
18 END FUNCTION zeroratio
```

ZeroRatio

3   Stopping Criteria

```fortran
LOGICAL PURE FUNCTION optimumstop(n, x,xp, iter, c) RESULT(stp)

INTEGER , INTENT(IN) :: iter, n
UREAL , INTENT(IN) :: x(n), xp(n), c(n)

INTEGER , PARAMETER :: iterlimit = 10000
UREAL :: obj, objp

stp = .FALSE.
obj = DOT(c,x)
objp = DOT(c,xp)

IF(iter >= iterlimit .OR. objp - obj < real(1.Q-100,realKind) ) stp = .TRUE.

END FUNCTION optimumstop
```

OptimumStop

```fortran
LOGICAL PURE FUNCTION potentialstop(n, x,xp, iter, c) RESULT(stp)

INTEGER , INTENT(IN) :: iter, n
UREAL , INTENT(IN) :: x(n), xp(n), c(n)

UREAL , PARAMETER :: delta= -one/8
INTEGER , PARAMETER :: iterlimit = 10000
UREAL ::  f, fp


stp = .FALSE.
fp = Potential(c,xp)
f =  Potential(c,x)

IF(iter >= iterlimit .OR. f - fp > delta) stp = .TRUE.

END FUNCTION potentialstop
```

PotentialStop

```fortran
LOGICAL PURE FUNCTION zerostop(n, x,xp, iter, c) RESULT(stp)

INTEGER , INTENT(IN) :: iter, n
UREAL , INTENT(IN) :: x(n), xp(n), c(n)

INTEGER , PARAMETER :: iterlimit = 10000


stp = .FALSE.

IF(iter >= iterlimit .OR. x(n-1) < real(1.Q-100,realKind) .OR. x(n-1) > xp(n-1)) stp = .TRUE.

END FUNCTION zerostop
```

ZeroStop

4 LPP Methods

4.1 Canonical

$$\text{minimize} \quad \mathbf{c}^T \mathbf{x}$$

$$\text{subject to} \begin{cases} \mathbf{Ax} = \mathbf{0} \\ \mathbf{x} \geq \mathbf{0} \quad \& \quad \sum_i x_i = 1 \end{cases}$$

```fortran
SUBROUTINE Canonical

UREAL :: Acan(row,column), ccan(column), xopt(column)
INTEGER :: i, j


OPEN(unit = 10, file= './1.IO/Acan.txt')
  DO i=1,row
    READ(10, *) (Acan(i,j) ,j=1,column)
  END DO
CLOSE(10)

OPEN(unit = 30, file= './1.IO/ccan.txt')
  DO j=1,column
    READ(30, *) ccan(j)
  END DO
CLOSE(30)


!CALL ProbGen(Acan)
!CALL ProbGen(ccan)

xopt = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)

END SUBROUTINE Canonical
```

Canonical

4.2 Equality

The Initial Problem is

$$\text{Find a feasible point} \quad \mathbf{x}_{feas}$$
$$\text{in Affine Space} \quad \mathbf{Ax} = \mathbf{b}$$
$$\text{s.t} \quad \mathbf{x} \geq \mathbf{0}$$

Introducing an artificial variable to create an interior starting point,

$$\text{Minimize} \quad \lambda$$
$$\text{in Affine Space} \quad \mathbf{Ax} + \lambda(\mathbf{b} - \mathbf{Ax}_0) = \mathbf{b}$$
$$\text{s.t} \quad \mathbf{x} \geq \mathbf{0}, \quad \text{for some } \mathbf{x}_0 \geq \mathbf{0}$$

This gives us an LPP in Equality form,

$$\text{Minimize} \quad \mathbf{c}_{eq}^T \mathbf{x}'$$
$$\text{in Affine Space} \quad \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq}$$
$$\text{s.t} \quad \mathbf{x}' \geq \mathbf{0}$$

where $\mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix}$, $\mathbf{b}_{eq} = \mathbf{b}$, $\mathbf{A}_{eq} = \begin{bmatrix} \mathbf{A} & \mathbf{b} - \mathbf{Ax}_0 \end{bmatrix}$ and $\mathbf{c}_{eq} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$

Projectively Transforming to Canonical form,

$$\text{Minimize} \quad \mathbf{c}_{can}^T \mathbf{x}''$$
$$\text{in Subspace} \quad \mathbf{A}_{can}\mathbf{x}'' = \mathbf{0}$$
$$\text{s.t} \quad \mathbf{x}'' \geq \mathbf{0}$$

```fortran
SUBROUTINE Equality

UREAL :: Aeq(row, column), beq(row), ceq(column), x0(column), xeq(column), eps, &
         Acan(size(Aeq,1),size(Aeq,2)+1), ccan(size(ceq)+1), xcan(size(Aeq,2)+1), &
         Acan1(size(Aeq,1),size(Aeq,2)+2), ccan1(size(ceq)+2), xcan1(size(Aeq,2)+2), &
            A(row,column+1), b(row), c(column+1), x01(column+1), xopt(column), xopt1( &
   column+1)
INTEGER :: i, j


OPEN(unit = 10, file= './1.IO/Aeq.txt')
DO i=1,row
READ(10, *) (Aeq(i,j) ,j=1,column)
END DO
CLOSE(10)

OPEN(unit = 20, file= './1.IO/beq.txt')
DO i=1,row
     READ(20, *) beq(i)
END DO
CLOSE(20)

OPEN(unit = 30, file= './1.IO/ceq.txt')
DO j=1,column
     READ(30, *) ceq(j)
END DO
CLOSE(30)

x0 = one

x01 = x0 .VAUG. one

b = beq + 0*(beq - GEMV(Aeq,x0))

A(:,:column) = Aeq
A(:,column+1) = 1*(beq - GEMV(Aeq,x0))

c(:column) = 0
c(column+1) = one

CALL ProjectiveTransform(A,b,c, Acan1,ccan1, x01)

!ccan = ccan - MAX(MAXVAL(ccan),0.)

xcan1 = ProjectiveScale(Acan1, ccan1, optimumstop, potentialratio)

xopt1 = InvProjTransform(xcan1, x01)

x0 = xopt1(:column)
print*, 'x0', xopt1
IF(x0(column+1) < 1.Q0-10) THEN

CALL ProjectiveTransform(Aeq,beq,ceq, Acan,ccan, x0)

xcan = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)

xopt = InvProjTransform(xcan, x0)
print*, 'xopt', xopt
END IF

END SUBROUTINE Equality
```

Equality

4.3   LeastNegative

Input

The Input LP Problem is in the Equality form:

$$\text{minimize} \quad \mathbf{c}^T \mathbf{x}$$

$$\text{subject to} \begin{cases} A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{cases}$$

Objective

To find the Least Negative lower bound to the system of equations $A\mathbf{x} = \mathbf{b}$ if exists, else find the Feasible point to the LP Problem.

Method

Step 1: Find a point satisfying the system of equations.

We use Least Squares Method to find a point $\mathbf{x}_0$ satisfying $A\mathbf{x} = \mathbf{b}$. The process is as follows:

$$AA^T \mathbf{y}_0 = \mathbf{b}$$
$$\mathbf{y}_0 = (AA^T)^{-1}\mathbf{b}$$
$$\mathbf{x}_0 = A^T \mathbf{y}_0$$

Step 2: Construct LPP to solve for lower bound of system of equations

Let

$$\varepsilon_0 = -2 * minval(\mathbf{x}_0)$$
$$\mathbf{x}_0' = \mathbf{x}_0 + \varepsilon_0$$

The point $\begin{bmatrix} \mathbf{x}_0' \\ \varepsilon_0 \end{bmatrix}$ satisfies the LP Problem:

$$\text{minimize} \quad \varepsilon$$

$$\begin{bmatrix} A & -rowsum(A) \end{bmatrix} \begin{bmatrix} \mathbf{x}' \\ \varepsilon \end{bmatrix} = \mathbf{b} \quad s.t \quad \begin{matrix} \mathbf{x}' \geq 0 \\ \varepsilon \geq 0 \end{matrix}$$

where $\mathbf{x}' = \mathbf{x} + \varepsilon$.

Step 3: Solve the LPP for zero Optimum

If the optimum solution of the LPP is zero, then the Feasible point for the Initial LPP can be found by $\mathbf{x}_0 = \mathbf{x}_{opt}'$, else ε_{opt} is the Least lower bound to the system of equations.

```fortran
1  SUBROUTINE LeastNegative
2
3  UREAL :: Aln(row, column), bln(row), cln(column), y0(row), x0(column), xln(column), eps,
       e(column), &
4               Aaug(row,column+1), baug(row), caug(column+1), x0aug(column+1), xaug(column
       +1), &
5                 Aaugcan(size(Aln,1),size(Aln,2)+2), caugcan(size(cln)+2), xaugcan(size(
       Aln,2)+2), &
6                 Acan(size(Aln,1),size(Aln,2)+1), ccan(size(cln)+1), xcan(size(Aln,2)+1)
7
8  INTEGER :: i, j
9
10
11 CALL ProbGen(Aln)
12 CALL ProbGen(bln)
13 CALL ProbGen(cln)
14
15 e=one
16
17 y0 = PDSOL(GEMM(Aln,TRANS(Aln)), bln)
18
19 x0 = GEMV(TRANS(Aln), y0)
20
21 IF(ANY(x0 < 0)) THEN
22
23 eps = -2*minval(x0)
24
25 Aaug = Aln .HAUG. -GEMV(Aln,e)
26
27 baug = bln
28
29 x0aug = (x0 + eps) .VAUG. eps
30
31 caug = 0
32 caug(column+1) = 1
33
34 CALL ProjectiveTransform(Aaug,baug,caug, Aaugcan, caugcan, x0aug)
35
36 xaugcan = ProjectiveScale(Aaugcan, caugcan, potentialstop, potentialratio)
37
38 xaug = InvProjTransform(xaugcan, x0aug)
39
40 x0 = xaug(:column)
41
42 END IF
43
44 IF(xaug(column+1) < 1.0Q-30) THEN
45
46 CALL ProjectiveTransform(Aln,bln,cln, Acan,ccan, x0)
47
48 ccan = ccan - MAX(MAXVAL(ccan),0.)
49
50 xcan = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)
51
52 xln = InvProjTransform(xcan, x0)
53
54 ELSE
55
56 eps = xaug(column +1)
57
58 END IF
59
60 END SUBROUTINE LeastNegative
```

LeastNegative

5 Modules

5.1 LPPGeneration

```fortran
#include "Preprocessor.F90"

#ifdef ProbDist
MODULE LPPGeneration
USE ProbabilityDistribution, ONLY : ProbDist
IMPLICIT NONE
! Generate LP Problem from Probability Distribution

PRIVATE
PUBLIC :: LPGen

CONTAINS

SUBROUTINE LPGen
! Generate Linear Programming Problem

UREAL :: A(row,column), b(row), c(column), x0(column)
INTEGER :: seed, count
INTEGER :: i, j


CALL SYSTEM_CLOCK(count)
seed = MOD(count,10000)

OPEN(unit = 10, file= './1.IO/A.txt')
DO i = 1, row
DO j = 1, column
    A(i,j) = ProbDist(seed)
    WRITE(10, *) A(i,j)
END DO
WRITE(10, *) ""
END DO
CLOSE(10)

OPEN(unit = 20, file= './1.IO/x0.txt')
DO i = 1, column
    x0(i) = ProbDist(seed)
    WRITE(20, *) x0(i)
END DO
CLOSE(20)

OPEN(unit = 30, file= './1.IO/c.txt')
DO i = 1, column
    c(i) = ProbDist(seed)
    WRITE(30, *) c(i)
END DO
CLOSE(30)

OPEN(unit = 40, file= './1.IO/b.txt')
DO i = 1, row
    b(i) = ProbDist(seed)
    WRITE(40, *) b(i)
END DO
CLOSE(40)

END SUBROUTINE LPGen

END MODULE LPPGeneration
#endif
```

LPPGeneration

5.2 ProbabilityGeneration

```fortran
#include "Preprocessor.F90"

#ifdef ProbDist
MODULE ProbabilityGeneration
USE ProbabilityDistribution , ONLY : ProbDist
IMPLICIT NONE
! Generate Probability Distribution vector/Matrix

PRIVATE
PUBLIC :: ProbGen

INTERFACE ProbGen

    MODULE PROCEDURE ProbVecGen
    MODULE PROCEDURE ProbMatGen

END INTERFACE

CONTAINS

SUBROUTINE ProbVecGen(v, trail)
!Generate Probablity Distribution Vector v integer trail

UREAL , INTENT(OUT) :: v(:)
INTEGER , OPTIONAL , INTENT(IN)  :: trail

INTEGER :: i, j, seed, count


IF(PRESENT(trail)) THEN
  seed = 12345
  DO j = 1, trail
    DO i = 1, size(v)
        v(i) = ProbDist(seed)
    END DO
  END DO
ELSE
  CALL SYSTEM_CLOCK(count)
  seed = MOD(count,10000)
END IF

DO i = 1, size(v)
    v(i) = ProbDist(seed)
END DO

END SUBROUTINE ProbVecGen

SUBROUTINE ProbMatGen(M, trail)
!Generate Probablity Distribution Matrix M with integer trail

UREAL , INTENT(OUT)              :: M(:,:)
INTEGER , OPTIONAL , INTENT(IN)  :: trail

INTEGER :: i, j, k, seed, count


IF(PRESENT(trail)) THEN
  seed = 12345
  DO k = 1, trail
    DO i = 1, size(M,1)
      DO j = 1, size(M,2)
        M(i,j) = ProbDist(seed)
      END DO
    END DO
  END DO
ELSE
```

```fortran
67     CALL SYSTEM_CLOCK(count)
68     seed = MOD(count,10000)
69  END IF
70
71  DO i = 1, size(M,1)
72     DO j = 1, size(M,2)
73        M(i,j) = ProbDist(seed)
74     END DO
75  END DO
76
77  END SUBROUTINE ProbMatGen
78
79  END MODULE ProbabilityGeneration
80  #endif
```

ProbabilityGeneration

5.3 ProbabilityDistributions

```fortran
1  #include "Preprocessor.F90"
2
3  #ifdef ProbDist
4  MODULE ProbabilityDistribution
5  IMPLICIT NONE
6
7  PRIVATE
8  PUBLIC :: ProbDist
9
10 CONTAINS
11
12 FUNCTION Uniform(seed) RESULT(u)
13 ! Uniform Distribution scalar u from integer seed
14
15 INTEGER :: seed
16 INTEGER, PARAMETER :: IA=16807,IM=2147483647,IQ=127773,IR=2836
17 UREAL, SAVE :: am
18 INTEGER, SAVE :: ix=-1,iy=-1,k
19
20 UREAL :: u
21
22
23 ! Initialise
24 IF (seed <= 0 .OR. iy < 0) THEN
25 am=nearest(1.0,-1.0)/IM
26 iy=ior(ieor(888889999,abs(seed)),1)
27 ix=ieor(777755555,abs(seed))
28 seed=abs(seed)+1
29 END IF
30
31 !!! Marsaglia shift sequence with period 2^32 -1
32 ix=ieor(ix,ishft(ix,13))
33 ix=ieor(ix,ishft(ix,-17))
34 ix=ieor(ix,ishft(ix,5))
35
36 ! Park-Miller sequence by Schrageâ ĂŹs method with period2^31âĹŠ2
37 k=iy/IQ
38 iy=IA*(iy-k*IQ)-IR*k
39 if (iy < 0) iy=iy+IM
40
41 ! Combine two Generators
42 u=am*ior(iand(IM,ieor(ix,iy)),1)
43
44 #ifdef lowerbound
45 #ifdef upperbound
46 u = (upperbound-lowerbound)*u + lowerbound
47 #endif
```

```fortran
48 #endif
49
50 END FUNCTION Uniform
51
52 FUNCTION Normal(seed) RESULT(n)
53 ! Normal Distribution scalar n from integer seed
54
55 INTEGER :: seed
56 UREAL, PARAMETER :: PI = 4.Q0*DATAN(1.D0)
57 UREAL :: u1,u2
58
59 UREAL :: n
60
61
62 u1 =  Uniform(seed)
63 u2 =  Uniform(seed)
64
65 n = sqrt(-2*log(u1))*cos(2*PI*u2)
66
67 #ifdef mean
68 #ifdef variance
69 n = variance*n + mean
70 #endif
71 #endif
72
73 END FUNCTION Normal
74
75 END MODULE ProbabilityDistribution
76 #endif
```

ProbabilityDistributions

5.4 LPPMethods

```fortran
1  #include "Preprocessor.F90"
2
3  MODULE LinearProblemSolver
4  USE ProbabilityGeneration, ONLY : ProbGen
5  USE ProjScalingAlgo, ONLY : ProjectiveScale
6  USE LPTools, ONLY : StdToCan, ProjectiveTransform, InvProjTransform, Dual, zerostop,
        potentialstop, optimumstop, zeroratio, minratio, potentialratio
7  USE LALibrary, ONLY : DOT, TRANS, GEMV, ENORM, PDSOL, GEMM
8  USE AUGOperator, ONLY : OPERATOR(.VAUG.), OPERATOR(.HAUG.)
9  IMPLICIT NONE
10
11 PRIVATE
12 PUBLIC :: LPType
13
14 CONTAINS
15
16 SUBROUTINE Canonical
17
18 UREAL :: Acan(row,column), ccan(column), xopt(column)
19 INTEGER :: i, j
20
21
22 OPEN(unit = 10, file= './1.IO/Acan.txt')
23   DO i=1,row
24     READ(10, *) (Acan(i,j) ,j=1,column)
25   END DO
26 CLOSE(10)
27
28 OPEN(unit = 30, file= './1.IO/ccan.txt')
29   DO j=1,column
30     READ(30, *) ccan(j)
31   END DO
```

```fortran
32  CLOSE(30)
33
34  !CALL ProbGen(c)
35
36  xopt = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)
37
38  END SUBROUTINE Canonical
39
40  SUBROUTINE Equality
41
42  UREAL :: Aeq(row, column), beq(row), ceq(column), x0(column), xeq(column), eps, &
43            Acan(size(Aeq,1),size(Aeq,2)+1), ccan(size(ceq)+1), xcan(size(Aeq,2)+1), &
44            Acan1(size(Aeq,1),size(Aeq,2)+2), ccan1(size(ceq)+2), xcan1(size(Aeq,2)+2), &
45            A(row,column+1), b(row), c(column+1), x01(column+1), xopt(column), xopt1(
       column+1)
46  INTEGER :: i, j
47
48
49  OPEN(unit = 10, file= './1.IO/Aeq.txt')
50  DO i=1,row
51  READ(10, *) (Aeq(i,j) ,j=1,column)
52  END DO
53  CLOSE(10)
54
55  OPEN(unit = 20, file= './1.IO/beq.txt')
56  DO i=1,row
57      READ(20, *) beq(i)
58  END DO
59  CLOSE(20)
60
61  OPEN(unit = 30, file= './1.IO/ceq.txt')
62  DO j=1,column
63      READ(30, *) ceq(j)
64  END DO
65  CLOSE(30)
66
67  x0 = one
68
69  x01 = x0 .VAUG. one
70
71  b = beq + 0*(beq - GEMV(Aeq,x0))
72
73  A(:,:column) = Aeq
74  A(:,column+1) = 1*(beq - GEMV(Aeq,x0))
75
76  c(:column) = 0
77  c(column+1) = one
78
79  CALL ProjectiveTransform(A,b,c, Acan1,ccan1, x01)
80
81  !ccan = ccan - MAX(MAXVAL(ccan),0.)
82
83  xcan1 = ProjectiveScale(Acan1, ccan1, optimumstop, potentialratio)
84
85  xopt1 = InvProjTransform(xcan1, x01)
86
87  x0 = xopt1(:column)
88  print*, 'x0', xopt1
89  IF(x0(column+1) < 1.Q0-10) THEN
90
91  CALL ProjectiveTransform(Aeq,beq,ceq, Acan,ccan, x0)
92
93  xcan = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)
94
95  xopt = InvProjTransform(xcan, x0)
96  print*, 'xopt', xopt
97  END IF
98
```

```fortran
99  END SUBROUTINE Equality

101 SUBROUTINE Standard

103 UREAL :: Astd(row,column), bstd(row), cstd(column), xopt(column), &
104                Acan(size(Astd,1)+size(Astd,2)+1,2*(size(Astd,1)+size(Astd,2)+1)), ccan(2*(
        size(Astd,1)+size(Astd,2)+1)), &
105                xcan(2*(size(Astd,1)+size(Astd,2)+1)), xopt1(2*(size(Astd,1)+size(Astd,2))
        +1), a0(2*(size(Astd,1)+size(Astd,2))+1)
106 INTEGER :: i, j


109 OPEN(unit = 10, file= './1.IO/Astd.txt')
110 DO i=1,row
111 READ(10, *) ( Astd(i,j) ,j=1,column)
112 END DO
113 CLOSE(10)

115 OPEN(unit = 40, file= './1.IO/bstd.txt')
116 DO i=1,row
117     READ(40, *) bstd(i)
118 END DO
119 CLOSE(40)

121 OPEN(unit = 30, file= './1.IO/cstd.txt')
122 DO j=1,column
123     READ(30, *) cstd(j)
124 END DO
125 CLOSE(30)

127 CALL StdtoCan(Astd,bstd,cstd, Acan,ccan, a0)

129 xcan = ProjectiveScale(Acan,ccan, potentialstop, potentialratio)

131 xopt1 = InvProjTransform(xcan, a0)

133 xopt = xopt1(:column)

135 END SUBROUTINE Standard

137 SUBROUTINE LeastNegative

139 UREAL :: Aln(row, column), bln(row), cln(column), y0(row), x0(column), xln(column), eps,
        e(column), &
140                Aaug(row,column+1), baug(row), caug(column+1), x0aug(column+1), xaug(column
        +1), &
141                Aaugcan(size(Aln,1),size(Aln,2)+2), caugcan(size(cln)+2), xaugcan(size(
        Aln,2)+2), &
142                Acan(size(Aln,1),size(Aln,2)+1), ccan(size(cln)+1), xcan(size(Aln,2)+1)

144 INTEGER :: i, j


147 OPEN(unit = 10, file= './1.IO/Aln.txt')
148 DO i=1,row
149 READ(10, *) (Aln(i,j) ,j=1,column)
150 END DO
151 CLOSE(10)

153 OPEN(unit = 20, file= './1.IO/bln.txt')
154 DO i=1,row
155     READ(20, *) bln(i)
156 END DO
157 CLOSE(20)

159 OPEN(unit = 30, file= './1.IO/cln.txt')
160 DO j=1,column+1
161     READ(30, *) cln(j)
```

```fortran
162  END DO
163  CLOSE (30)
164
165  e=one
166
167  y0 = PDSOL(GEMM(Aln,TRANS(Aln)), bln)
168
169  x0 = GEMV(TRANS(Aln), y0)
170
171  IF(ANY(x0 < 0)) THEN
172
173  eps = -2*minval(x0)
174
175  Aaug = Aln .HAUG. -GEMV(Aln,e)
176
177  baug = bln
178
179  x0aug = (x0 + eps) .VAUG. eps
180
181  caug = 0
182  caug(column+1) = 1
183
184  CALL ProjectiveTransform(Aaug,baug,caug, Aaugcan, caugcan, x0aug)
185
186  xaugcan = ProjectiveScale(Aaugcan, caugcan, potentialstop, potentialratio)
187
188  xaug = InvProjTransform(xaugcan, x0aug)
189
190  x0 = xaug(:column)
191
192  END IF
193
194  IF(xaug(column+1) < 1.0Q-30) THEN
195
196  CALL ProjectiveTransform(Aln,bln,cln, Acan,ccan, x0)
197
198  ccan = ccan - MAX(MAXVAL(ccan),0.)
199
200  xcan = ProjectiveScale(Acan, ccan, potentialstop, potentialratio)
201
202  xln = InvProjTransform(xcan, x0)
203
204  ELSE
205
206  eps = xaug(column +1)
207
208  END IF
209
210  END SUBROUTINE LeastNegative
211
212  END MODULE LinearProblemSolver
```

LPPMethods

5.5 ProjScalingAlgo

```fortran
 1  #include "Preprocessor.F90"
 2
 3  MODULE ProjScalingAlgo
 4  USE AUGOperator, ONLY : OPERATOR(.VAUG.)
 5  USE LALibrary, ONLY : DOT, ENORM, GEMV, GEMM, TRANS, PDSOL, COLMULT
 6  IMPLICIT NONE
 7
 8
 9  PUBLIC :: ProjectiveScale
10  PRIVATE
```

```fortran
11
12 CONTAINS
13
14
15 PURE FUNCTION ProjectiveScale(A,c, stopcond, ratiotest) RESULT(xcan)
16 ! Obtain vector xcan by minimizing along Objective vector c of system Axcan = 0 using
      stoping condition stopcond and ratio test ratiotest functions
17
18 UREAL, INTENT(IN) :: A(:,:), c(:)
19 UREAL             :: xcan(size(A,2))
20
21 UREAL :: xp(size(A,2)), x(size(A,2))
22 INTEGER :: n, iter
23
24 INTERFACE
25
26   UREAL PURE FUNCTION ratiotest(n, cunit) RESULT(alpha)
27     INTEGER, INTENT(IN) :: n
28     UREAL,   INTENT(IN) :: cunit(n)
29     END FUNCTION ratiotest
30
31     LOGICAL PURE FUNCTION stopcond(n, x,xp, iter, c) RESULT(stp)
32   INTEGER, INTENT(IN) :: iter, n
33   UREAL, INTENT(IN) :: x(n), xp(n), c(n)
34   END FUNCTION stopcond
35
36 END INTERFACE
37
38
39 !IF(size(A,2) /= size(c)) STOP "Algorithm ERROR: Wrong size for input argument"
40
41 n = size(A,2)
42 xp = one/n
43
44 !IF( ANY(GEMV(A,xp) >= ukind(1.Q-10)) ) STOP "Center of Simplex doesn't lie in Null space
       of the Input Matrix:"
45
46 iter = 1
47 DO
48
49 x = Optimize(xp)
50 IF(stopcond(n, x,xp, iter, c)) EXIT
51
52 !IF( ANY(ISNAN(x))) THEN
53 !PRINT*,  "WARNING: NAN occurred in the Solution"
54 !x= xp
55 !EXIT
56 !END IF
57
58 xp = x
59 iter = iter + 1
60
61 END DO
62
63 xcan = x
64
65 CONTAINS
66
67 PURE FUNCTION Optimize(xp) RESULT(x)
68
69 UREAL, INTENT(IN) :: xp(:)
70 UREAL             :: x(size(xp))
71
72 UREAL :: Ad(size(A,1),size(xp)), B(size(A,1)+1,size(xp)), &
73            v(size(A,1)+1), cp(size(c)), cunit(size(c)), alpha
74 UREAL ::  e(size(xp)) = one
75 UREAL :: x0(size(xp)) = e/n
76
```

```
77
78 Ad = COLMULT(xp,A)
79 B = Ad .VAUG. e
80 v = PDSOL(GEMM(B,TRANS(B)), GEMV(B,xp*c))
81
82 cp = xp*c - GEMV(TRANS(B),v)
83 cunit = cp/ENORM(cp)
84
85 alpha = ratiotest(n, cunit)
86
87 x = x0 - alpha*cunit
88 x = x*xp/DOT(x,xp)
89
90 END FUNCTION Optimize
91
92 END FUNCTION ProjectiveScale
93
94
95 END MODULE ProjScalingAlgo
```

ProjScalingAlgo

5.6 LALibrary

```
1 #include "Preprocessor.F90"
2
3 MODULE LALibrary
4 IMPLICIT NONE
5 !!! Linear Algebra Library
6
7
8 PRIVATE
9 PUBLIC :: DIAG, ONES, COLMULT, DOT, ENORM, GEMV, GEMM, PDSOL, TRANS, NSQRT, NLOG, ADD
10
11 CONTAINS
12
13 PURE FUNCTION DIAG(x) RESULT(D)
14 ! Diagnoal Matrix D of vector x
15
16 UREAL, INTENT(IN) :: x(:)
17 UREAL            :: D(size(x), size(x))
18
19 INTEGER :: i
20
21
22 D=0
23 DO i = 1,size(x)
24 D(i,i) = x(i)
25 END DO
26
27 END FUNCTION DIAG
28
29 PURE FUNCTION ONES(n) RESULT(D)
30 ! Diagnol Matrix D of integer dimension n
31
32 INTEGER, INTENT(IN) :: n
33 UREAL            :: D(n,n)
34
35 INTEGER :: i
36
37
38 D=0
39 DO i = 1,n
40 D(i,i) = 1
41 END DO
42
```

```fortran
43 END FUNCTION ONES

44
45 PURE FUNCTION COLMULT(u,A) RESULT(uA)
46 ! Multiply Columns of Matrix A with vector u to get Matrix uA

47
48 UREAL, INTENT(IN) :: u(:),A(:,:)
49 UREAL            :: uA(size(A,1),size(A,2))

50
51 INTEGER :: i

52
53
54 DO i = 1,size(u)
55     uA(:,i) = u(i)*A(:,i)
56 END DO

57
58 END FUNCTION COLMULT

59
60
61 PURE FUNCTION ADD(v) RESULT(a)
62 ! Add all the elements of vector v to get a scalar a

63
64 UREAL, INTENT(IN) :: v(:)
65 UREAL            :: a

66
67 INTEGER :: i

68
69
70 a = 0
71 DO i = 1, size(v)
72   a = a + v(i)
73 END DO

74
75 END FUNCTION ADD

76
77
78 PURE FUNCTION DOT(u,v) RESULT(uTv)
79 ! Dot Product vectors u and v t get a scalar uTv

80
81 UREAL, INTENT(IN) :: u(:),v(:)
82 UREAL            :: uTv

83
84
85 uTv = ADD(u(:)*v(:))

86
87 END FUNCTION DOT

88
89 ELEMENTAL FUNCTION NSQRT(n) RESULT(s)
90 ! Square root s of n

91
92 UREAL, INTENT(IN) :: n
93 UREAL            :: s

94
95 INTEGER :: i

96
97
98 !IF(n<0) STOP "NSQRT ERROR: Invalid Input"

99
100 s = n/2

101
102 DO i = 1,realKind
103   s = (s + n/s)/2
104 END DO

105
106 END FUNCTION NSQRT

107
108 ELEMENTAL FUNCTION NLOG(n) RESULT(l)
109 ! Natural Logarithm l of n
110
```

```fortran
111 UREAL , INTENT ( IN ) :: n
112 UREAL             :: l
113
114
115 l = 2* ATANH (( n -1)/( n +1))
116
117 END FUNCTION NLOG
118
119 PURE FUNCTION ENORM ( v ) RESULT ( n )
120 ! Euclidean Norm scalar n of vector v
121
122 UREAL , INTENT ( IN ) :: v (:)
123 UREAL             :: n
124
125 n = NSQRT ( DOT (v , v ))
126
127 END FUNCTION ENORM
128
129 PURE FUNCTION CholeskyDecomposition ( A ) RESULT ( L )
130 ! Lower Triangular Matrix L by Cholesky Decomposition of Matrix A
131
132 UREAL , INTENT ( IN ) :: A (: ,:)
133 UREAL             :: L ( size (A ,1) , size (A ,2))
134
135 INTEGER :: i
136 UREAL    :: B ( size (A ,1) , size (A ,2)) , summ
137
138
139 ! IF ( size (A ,1) /= size (A ,2)) STOP " CHOLESKY ERROR : Invalid size "
140 B = A
141 L = 0
142
143 DO i = 1 , size (B ,1)
144 summ = B (i , i ) - DOT ( B (i ,: i -1) , B (i ,: i -1))
145 ! IF ( summ <= 0.) STOP " CHOLESKY ERROR : Invalid Matrix Input "
146 L (i , i ) = NSQRT ( summ )
147 B ( i +1: , i )=( B (i , i +1:) - GEMV ( B ( i +1: ,: i -1) , B (i ,: i -1)))/ L (i , i )
148 L ( i +1: , i ) = B ( i +1: , i )
149 END DO
150
151 END FUNCTION CholeskyDecomposition
152
153 PURE FUNCTION ForSubstitution (L , u ) RESULT ( v )
154 ! vector v by Forward Substitution of Lower Triangular Matrix L and vector u
155
156 UREAL , INTENT ( IN ) :: L (: ,:) , u (:)
157 UREAL             :: v ( size (u ,1))
158
159 INTEGER :: i
160
161
162 ! IF ( size (L ,2) /= size ( u )) STOP " BackSubstitution ERROR : Invalid size input "
163
164 DO i   = 1 , size (u ,1)
165 v ( i ) = ( u ( i ) - ADD ( L (i ,: i -1)* v (: i -1)))/ L (i , i )
166 END DO
167
168 END FUNCTION ForSubstitution
169
170 PURE FUNCTION BackSubstitution (U , a ) RESULT ( v )
171 ! vector v by Forward Backward of Upper Triangular Matrix U and vector a
172
173 UREAL , INTENT ( IN ) :: U (: ,:) , a (:)
174 UREAL             :: v ( size (a ,1))
175
176 INTEGER :: i
177
178
```

```fortran
179 !IF(size(U,2) /= size(b)) STOP "ForSubstitution ERROR: Invalid size input"
180
181 DO i  = size(a,1),1,-1
182 v(i) = (a(i) - SUM(U(i,i+1:)*v(i+1:)))/U(i,i)
183 END DO
184
185 END FUNCTION BackSubstitution
186
187
188 PURE FUNCTION GEMV(M,v) RESULT(Mv)
189 ! vector Mv by Matrix Vector Multiplication of Matrix M and vector v
190
191 UREAL , INTENT(IN) :: M(:,:),v(:)
192 UREAL            :: Mv(size(M,1))
193
194 INTEGER :: i
195
196
197 !IF(size(M,2) /= size(v)) STOP "MatVecMult ERROR: Invalid size input"
198
199 DO i = 1,size(M,1)
200     Mv(i) = DOT(M(i,:),v(:))
201 END DO
202
203 END FUNCTION GEMV
204
205 PURE FUNCTION GEMM(A,B) RESULT(AB)
206 ! Matrix AB by Matrix Vector Multiplication of Matrix A and Matrix B
207
208 UREAL , INTENT(IN) :: A(:,:),B(:,:)
209 UREAL            :: AB(size(A,1),size(B,2))
210
211 INTEGER :: i, j
212
213
214 DO i = 1,size(A,1)
215     DO j = 1,size(B,2)
216         AB(i,j) = DOT(A(i,:),B(:,j))
217     END DO
218 END DO
219
220 END FUNCTION GEMM
221
222 PURE FUNCTION PDSOL(A, y) RESULT(x)
223 !vector x by solving System Ax = y of Positive Definite Symmetric Matrix A and vector y
224
225 UREAL , INTENT(IN) :: A(:,:), y(:)
226 UREAL            :: x(size(A,2))
227
228 UREAL :: L(size(A,1), size(A,1)), u(size(A,1))
229
230
231 !IF(size(A, 1) /= size(A, 2) .OR. size(A,1) /= size(y)) STOP "PDLSS ERROR: Invalid Input"
232
233 L = CholeskyDecomposition(A)
234
235 u = ForSubstitution(L, y)
236
237 x = BackSubstitution(TRANS(L), u)
238
239 END FUNCTION PDSOL
240
241 PURE FUNCTION TRANS(A) RESULT(AT)
242 ! Transpose AT of Matrix A
243
244 UREAL , INTENT(IN) :: A(:,:)
245 UREAL            :: AT(size(A,2),size(A,1))
246
```

```
247  INTEGER :: i, j
248
249
250  FORALL(i = 1:size(A,1) , j = 1:size(A,2))
251      AT(j,i) = A(i,j)
252  END FORALL
253
254  END FUNCTION TRANS
255
256  END MODULE LALibrary
```

LALibrary

5.7 LPTools

```
1   #include "Preprocessor.F90"
2
3   MODULE LPTools
4   USE AUGOperator , ONLY : OPERATOR(.VAUG.)
5   USE ProbabilityGeneration , ONLY : ProbGen
6   USE LALibrary , ONLY : ONES, DOT, ColMult, GEMV, ENORM, TRANS, NLOG, ADD, NSQRT
7   IMPLICIT NONE
8
9   PRIVATE
10  PUBLIC :: Dual , StdToCan , ProjectiveTransform , InvProjTransform , zerostop , potentialstop ,
         optimumstop , zeroratio , minratio , potentialratio
11
12  CONTAINS
13
14
15  UREAL PURE FUNCTION Potential(c,x) RESULT(f)
16
17  UREAL , INTENT(IN) :: c(column+1) , x(column+1)
18
19
20  f = ADD(NLOG(DOT(c,x)/x))
21
22  END FUNCTION Potential
23
24  LOGICAL PURE FUNCTION zerostop(n, x,xp, iter, c) RESULT(stp)
25
26  INTEGER , INTENT(IN) :: iter, n
27  UREAL , INTENT(IN) :: x(n), xp(n), c(n)
28
29  INTEGER , PARAMETER :: iterlimit = 10000
30
31
32  stp = .FALSE.
33
34  IF(iter >= iterlimit .OR. x(n-1) < real(1.Q-100,realKind) .OR. x(n-1) > xp(n-1)) stp = .
         TRUE.
35
36  END FUNCTION zerostop
37
38  LOGICAL PURE FUNCTION potentialstop(n, x,xp, iter, c) RESULT(stp)
39
40  INTEGER , INTENT(IN) :: iter, n
41  UREAL , INTENT(IN) :: x(n), xp(n), c(n)
42
43  UREAL , PARAMETER :: delta= -one/8
44  INTEGER , PARAMETER :: iterlimit = 10000
45  UREAL ::  f, fp
46
47
48  stp = .FALSE.
49  fp = Potential(c,xp)
```

24

```fortran
50 f =   Potential(c,x)
51
52 IF(iter >= iterlimit .OR. f - fp > delta) stp = .TRUE.
53
54 END FUNCTION potentialstop
55
56 LOGICAL PURE FUNCTION optimumstop(n, x,xp, iter, c) RESULT(stp)
57
58 INTEGER, INTENT(IN) :: iter, n
59 UREAL, INTENT(IN) :: x(n), xp(n), c(n)
60
61 INTEGER, PARAMETER :: iterlimit = 10000
62 UREAL :: obj, objp
63
64 stp = .FALSE.
65 obj = DOT(c,x)
66 objp = DOT(c,xp)
67
68 IF(iter >= iterlimit .OR. objp - obj < real(1.Q-100,realKind) ) stp = .TRUE.
69
70 END FUNCTION optimumstop
71
72 UREAL PURE FUNCTION potentialratio(n, cunit) RESULT(alpha)
73
74 INTEGER, INTENT(IN) :: n
75 UREAL, INTENT(IN) :: cunit(n)
76
77
78 alpha = 1/(4*NSQRT(ukind(n)*(ukind(n)-1)))
79
80 END FUNCTION potentialratio
81
82 UREAL PURE FUNCTION zeroratio(n, cunit) RESULT(alpha)
83
84 INTEGER, INTENT(IN) :: n
85 UREAL, INTENT(IN) :: cunit(n)
86
87 UREAL, PARAMETER :: beta = ukind(1.Q-1)
88 UREAL :: a
89 INTEGER :: idx
90
91
92 alpha = one/(n*cunit(n-1))
93 DO idx=1,n
94   IF(cunit(idx)<0 .OR. idx == n-1) CYCLE
95   a = (ukind(1.Q0)-beta)/(n*cunit(idx))
96   IF(alpha>a) alpha = a
97 END DO
98
99 END FUNCTION zeroratio
100
101 UREAL PURE FUNCTION minratio(n, cunit) RESULT(alpha)
102
103 INTEGER, INTENT(IN) :: n
104 UREAL, INTENT(IN) :: cunit(n)
105
106 UREAL, PARAMETER :: beta = ukind(1.Q-1)
107
108
109 alpha = (one-beta)/(n*maxval(cunit))
110
111 END FUNCTION minratio
112
113
114 PURE SUBROUTINE Dual(A, b, c, Adual, bdual, cdual)
115
116 UREAL, INTENT(IN) :: A(:,:), b(:), c(:)
117
```

```fortran
118  UREAL , INTENT ( OUT ) :: Adual ( size (A ,2) , size (A ,1) ) , bdual ( size (c) ) , cdual ( size (b) )
119
120
121  cdual = -b
122
123  Adual = - TRANS (A)
124
125  bdual = -c
126
127  END SUBROUTINE Dual
128
129  PURE SUBROUTINE StdtoCan ( Astd , bstd , cstd ,Acan , ccan , a0)
130
131  UREAL , INTENT (IN) :: Astd (: ,:) , bstd (:) , cstd (:)
132  UREAL , INTENT ( OUT ) :: Acan ( size ( Astd ,1) + size ( Astd ,2) +1 ,2*( size ( Astd ,1) + size ( Astd ,2) +1) ) ,
        &
133                   ccan (2*( size ( Astd ,1) + size ( Astd ,2) +1) ) , a0 (2*( size ( cstd ) + size ( bstd ) ) +1)
134
135  UREAL :: A( size ( Astd ,1) + size ( Astd ,2) +1 ,2*( size ( Astd ,1) + size ( Astd ,2) ) +1) , &
136              b( size ( Astd ,1) + size ( Astd ,2) +1) ,c (2*( size ( Astd ,1) + size ( Astd ,2) ) +1)
137  UREAL :: x0( size ( Astd ,2) ) , y0( size ( Astd ,1) ) , u0( size ( Astd ,1) ) , v0( size ( Astd ,2) ) , lambda0
138
139  INTEGER :: m, n, i,j
140
141
142  m = size ( Astd ,1) ; n = size ( Astd ,2)
143
144  x0 = 1
145  y0 = 1
146  u0 = 1
147  v0 = 1
148  lambda0 = 1
149
150  a0 = x0 .VAUG. y0 .VAUG. u0 .VAUG. v0 .VAUG. lambda0
151
152  A = 0
153
154  A (1:m ,1:n) = Astd
155  A (1:m ,n+1:n+m) = -Ones (m)
156
157  A(m+1:m+n ,m+n+1:2*m+n) = TRANS ( Astd )
158  A(m+1:m+n ,2*m+n+1:2*(m+n) ) = Ones (n)
159
160  A(m+n+1 ,1:n) = cstd
161  A(m+n+1 ,m+n+1:2*m+n) = -bstd
162
163  A (1:m ,2*(m+n) +1) =  bstd - GEMV ( Astd ,x0) + y0
164  A(m+1:m+n ,2*(m+n) +1) = cstd - GEMV ( TRANS ( Astd ) ,u0) - v0
165  A(m+n+1 ,2*(m+n) +1) = -DOT ( cstd ,x0) + DOT ( bstd ,u0)
166
167
168  b (:m) = bstd
169  b(m+1:m+n) = cstd
170  b(m+n+1) = 0
171
172
173  c = 0
174  c (2*m+2*n+1) = 1
175
176  CALL ProjectiveTransform (A,b,c, Acan , ccan , a0)
177
178  END SUBROUTINE StdToCan
179
180  PURE SUBROUTINE ProjectiveTransform (A,b,c, Acan , ccan , a0)
181
182  UREAL , INTENT (IN) :: A(: ,:) ,b(:) ,c(:) , a0 (:)
183
184  UREAL , INTENT ( OUT ) :: Acan ( size (A ,1) , size (A ,2) +1) , ccan ( size (c) +1)
```

```fortran
185
186
187 Acan(:,:size(A,2)) = ColMult(a0,A)
188 Acan(:,size(A,2)+1) = -b
189
190 ccan(:size(c)) = a0*c
191 ccan(size(c)+1) = 0
192
193 END SUBROUTINE ProjectiveTransform
194
195 PURE FUNCTION InvProjTransform(xcan,x0) RESULT(x)
196
197 UREAL, INTENT(IN) :: xcan(:), x0(:)
198
199 UREAL             :: x(size(xcan)-1)
200
201
202 x = (x0*xcan(:size(x0)))/xcan(size(xcan))
203
204 END FUNCTION InvProjTransform
205
206 END MODULE LPTools
```

LPTools

5.8 AUGOperator

```fortran
 1 #include "Preprocessor.F90"
 2
 3 MODULE AUGOperator
 4 IMPLICIT NONE
 5 !!! AUGMENT OPERATION on ANY SCALAR/VECTOR/MATRIX AND ANY SCALAR/VECTOR/MATRIX
 6
 7
 8 PRIVATE :: sHAugv, vHaugs, vHAugv, MHAugs, sHAugM, MHAugv, vHAugM, MHAugM, sVAugs, sVAugv
           , vVAugs, vVAugv, vVAugM, MVAugv, MVAugM
 9
10 PUBLIC  :: OPERATOR(.HAUG.), OPERATOR(.VAUG.)
11 ! .HAUG. Operates Horizontal Augmentation and .VAUG. Operates Vertical Augmentation
12
13
14 INTERFACE OPERATOR(.HAUG.)
15   MODULE PROCEDURE sHAugv
16   MODULE PROCEDURE vHAugs
17   MODULE PROCEDURE vHAugv
18   MODULE PROCEDURE MHAugs
19   MODULE PROCEDURE sHAugM
20   MODULE PROCEDURE MHAugv
21   MODULE PROCEDURE vHAugM
22   MODULE PROCEDURE MHAugM
23 END INTERFACE
24
25 INTERFACE OPERATOR(.VAUG.)
26   MODULE PROCEDURE sVAugs
27   MODULE PROCEDURE sVAugv
28   MODULE PROCEDURE vVAugs
29   MODULE PROCEDURE vVAugv
30   MODULE PROCEDURE vVAugM
31     MODULE PROCEDURE MVAugv
32     MODULE PROCEDURE MVAugM
33 END INTERFACE
34
35
36 CONTAINS
37
38
```

```fortran
39  PURE FUNCTION sHAugv(s,v) RESULT(sv)
40  ! Horizontal Augment SCALAR and VECTOR : sv = [s v]
41
42  UREAL , INTENT(IN) :: s, v(:)
43  UREAL             :: sv(size(v),2)
44
45  sv = 0
46  sv(1,1) = s
47  sv(:,2) = v(:)
48
49  END FUNCTION sHAugv
50
51  PURE FUNCTION vHAugs(v,s) RESULT(vs)
52  ! Horizontal Augment VECTOR and SCALAR : vs = [v s]
53
54  UREAL , INTENT(IN) :: v(:), s
55  UREAL             :: vs(size(v),2)
56
57
58  vs = 0
59  vs(:,1) = v(:)
60  vs(1,2) = s
61
62  END FUNCTION vHAugs
63
64  PURE FUNCTION vHAugv(u,v) RESULT(uv)
65  ! Horizontal Augment VECTOR and VECTOR : uv = [u v]
66
67  UREAL , INTENT(IN) :: u(:), v(:)
68  UREAL             :: uv(max(size(u),size(v)),2)
69
70
71  uv = 0
72  uv(:size(u),1) = u(:)
73  uv(:size(v),2) = v(:)
74
75  END FUNCTION vHAugv
76
77  PURE FUNCTION MHAugs(M,s) RESULT(Ms)
78  ! Horizontal Augment MATRIX and SCALAR : Ms = [M s]
79
80  UREAL , INTENT(IN) :: M(:,:), s
81  UREAL             :: Ms(size(M,1),size(M,2)+1)
82
83
84  Ms = 0
85  Ms(:,:size(M,2)) = M
86  Ms(1,size(M,2)+1) = s
87
88  END FUNCTION MHAugs
89
90  PURE FUNCTION sHAugM(s,M) RESULT(sM)
91  ! Horizontal Augment SCALAR and MATRIX : sM = [s M]
92
93  UREAL , INTENT(IN) :: s, M(:,:)
94  UREAL             :: sM(size(M,1),size(M,2)+1)
95
96
97  sM = 0
98  sM(1,1) = s
99  sM(:,2:) = M
100
101 END FUNCTION sHAugM
102
103 PURE FUNCTION MHAugv(M,v) RESULT(Mv)
104 ! Horizontal Augment MATRIX and VECTOR : Mv = [M v]
105
106 UREAL , INTENT(IN) :: M(:,:), v(:)
```

```fortran
107 UREAL                   :: Mv(max(size(M,1),size(v)),size(M,2)+1)
108
109
110 Mv = 0
111 Mv(:size(M,1),:size(M,2)) = M
112 Mv(:size(v),size(M,2)+1) = v
113
114 END FUNCTION MHAugv
115
116 PURE FUNCTION vHAugM(v,M) RESULT(vM)
117 ! Horizontal Augment VECTOR and MATRIX : vM = [v M]
118
119 UREAL, INTENT(IN) :: v(:), M(:,:)
120 UREAL                   :: vM(max(size(v),size(M,1)),size(M,2)+1)
121
122
123 vM = 0
124 vM(:size(M),1) = v
125 vM(:size(M,1),2:) = M
126
127 END FUNCTION vHAugM
128
129 PURE FUNCTION MHAugM(M,N) RESULT(MN)
130 ! Horizontal Augment MATRIX and MATRIX : MN = [M N]
131
132 UREAL, INTENT(IN) :: M(:,:), N(:,:)
133 UREAL                   :: MN(max(size(M,1),size(N,1)),size(M,2)+size(N,2))
134
135
136 MN = 0
137 MN(:size(M,1),:size(M,2)) = M
138 MN(:size(N,1),size(M,2)+1:size(M,2)+size(N,2)) = N
139
140 END FUNCTION MHAugM
141
142
143 PURE FUNCTION sVAugs(s,t) RESULT(st)
144 ! Vertical Augment SCALAR and SCALAR : st = [s t]^T
145
146 UREAL, INTENT(IN) :: s,t
147 UREAL             :: st(2)
148
149
150 st(1) = s
151 st(2) = t
152
153 END FUNCTION sVAugs
154
155 PURE FUNCTION sVAugv(s,v) RESULT(sv)
156 ! Vertical Augment SCALAR and VECTOR : sv = [s v]^T
157
158 UREAL, INTENT(IN) :: s, v(:)
159 UREAL             :: sv(1+size(v))
160
161
162 sv(1) = s
163 sv(2:) = v
164
165 END FUNCTION sVAugv
166
167 PURE FUNCTION vVAugs(v,s) RESULT(vs)
168 ! Vertical Augment VECTOR and SCALAR : vs = [v s]^T
169
170 UREAL, INTENT(IN) :: v(:), s
171 UREAL             :: vs(size(v)+1)
172
173
174 vs(:size(v)) = v
```

```fortran
175 vs(size(v)+1) = s
176
177 END FUNCTION vVAugs
178
179
180 PURE FUNCTION vVAugv(u,v) RESULT(uv)
181 ! Vertical Augment VECTOR and VECTOR : uv = [u v]^T
182
183 UREAL, INTENT(IN) :: u(:), v(:)
184 UREAL            :: uv(size(u)+size(v))
185
186
187 uv(:size(u)) = u
188 uv(size(u)+1:) = v
189
190 END FUNCTION vVAugv
191
192 PURE FUNCTION vVAugM(v,M) RESULT(vM)
193 ! Vertical Augment VECTOR and MATRIX : vM = [v^T M]^T
194
195 UREAL, INTENT(IN) :: v(:), M(:,:)
196 UREAL :: vM(size(M,1)+1,max(size(v),size(M,2)))
197
198
199 vM = 0
200 vM(1,:size(v)) = v
201 vM(2: ,:size(M,2)) = M
202
203 END FUNCTION vVAugM
204
205 PURE FUNCTION MVAugv(M,v) RESULT(Mv)
206 ! Vertical Augment VECTOR and MATRIX : Mv = [M v^T]^T
207
208 UREAL, INTENT(IN) :: M(:,:), v(:)
209 UREAL :: Mv(size(M,1)+1,max(size(M,2),size(v)))
210
211
212 Mv = 0
213 Mv(:,:size(M,2)) = M
214 Mv(size(M,1)+1,:size(v)) = v
215
216 END FUNCTION MVAugv
217
218 PURE FUNCTION MVAugM(M,N) RESULT(MN)
219 ! Vertical Augment VECTOR and MATRIX : MN = [M N]^T
220
221 UREAL, INTENT(IN) :: M(:,:), N(:,:)
222 UREAL :: MN(size(M,1)+size(N,1),max(size(M,2),size(N,2)))
223
224
225 MN = 0
226 MN(:size(M,1),:size(M,2)) = M
227 MN(size(M,1)+1:size(M,1)+size(N,1),:size(N,2)) = N
228
229 END FUNCTION MVAugM
230
231 END MODULE AUGOperator
```

AUGOperator