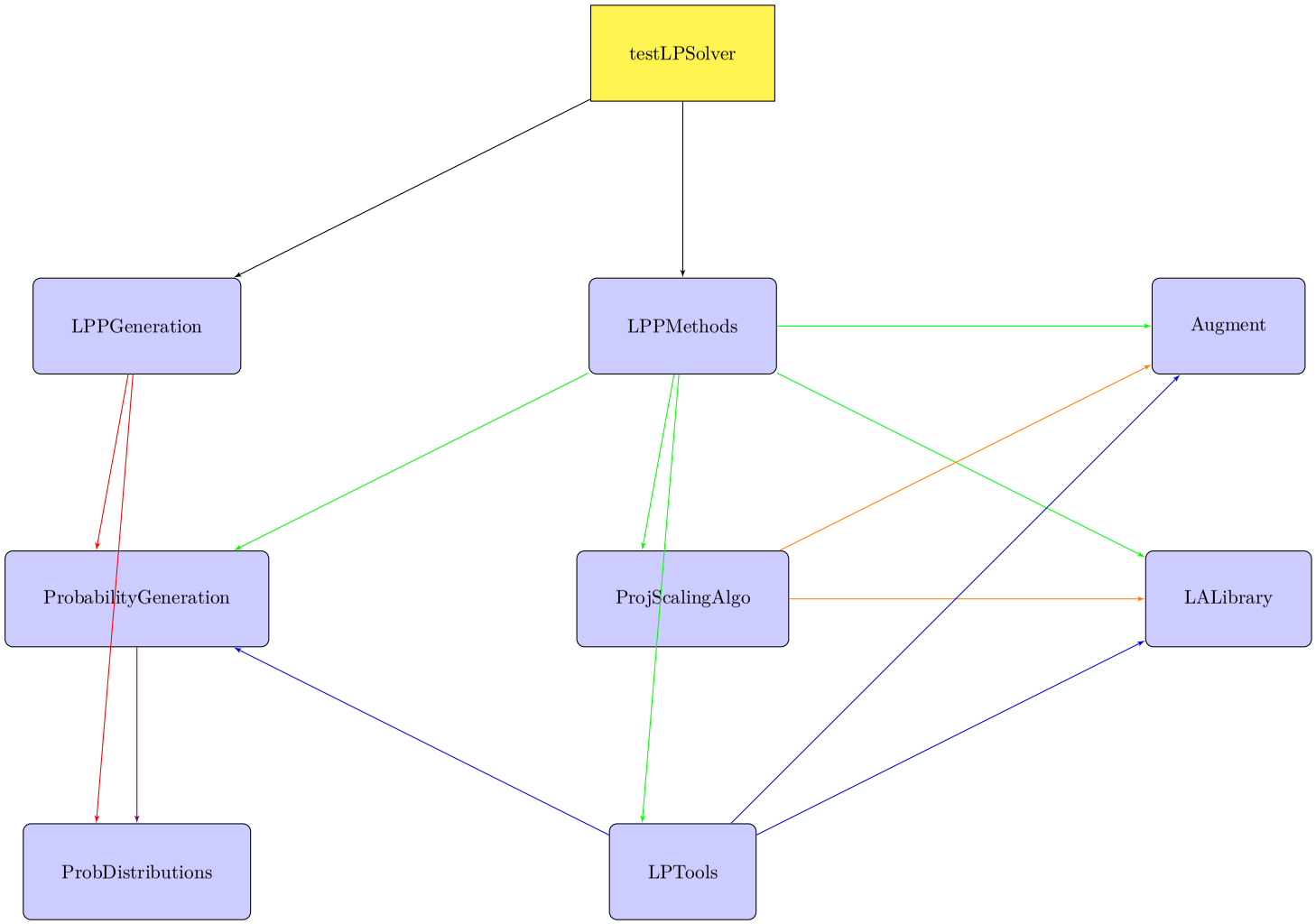
LP Solver

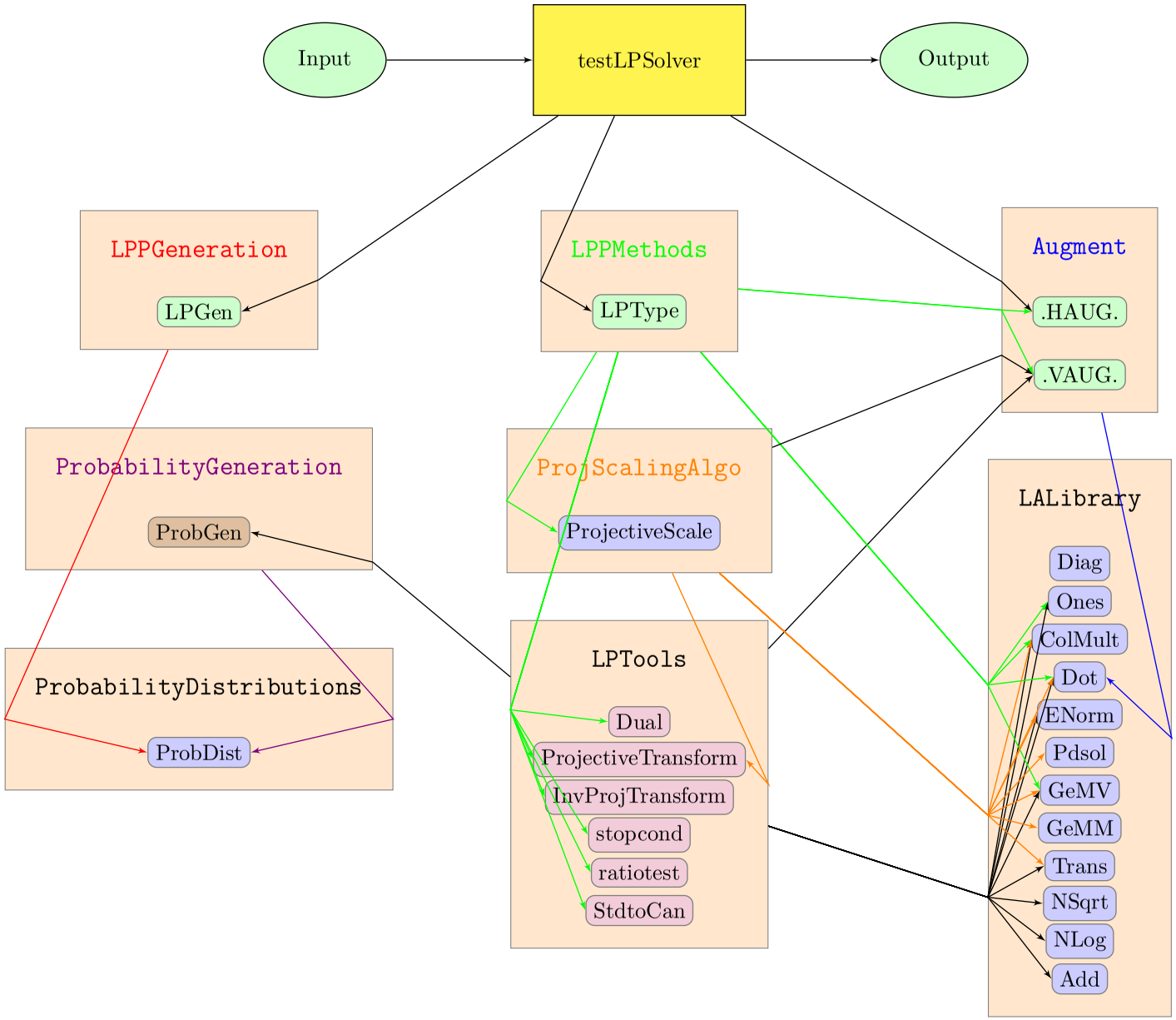
Akhil Akkapelli

December 16, 2021

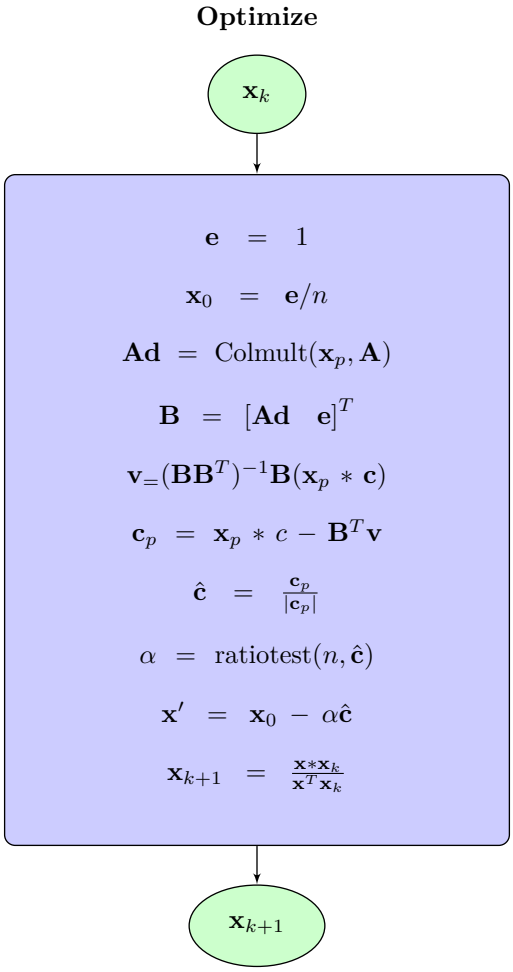
0.1 Simple Module Dependency Diagram



0.2 Detailed Modular Dependency Diagram



1 Projective Scaling Algorithm




```
PURE FUNCTION ProjectiveScale(A,c, stopcond, ratiotest) RESULT(xcan)
! Obtain vector xcan by minimizing along Objective vector c of system Axcan = 0 using
    stoping condition stopcond and ratio test ratiotest functions

UREAL, INTENT(IN) :: A(:,:), c(:)
UREAL             :: xcan(size(A,2))

UREAL :: xp(size(A,2)), x(size(A,2))
INTEGER :: n, iter

INTERFACE

  UREAL PURE FUNCTION ratiotest(n, cunit) RESULT(alpha)
    INTEGER, INTENT(IN) :: n
    UREAL,   INTENT(IN) :: cunit(n)
    END FUNCTION ratiotest

    LOGICAL PURE FUNCTION stopcond(n, x,xp, iter, c) RESULT(stp)
  INTEGER, INTENT(IN) :: iter, n
  UREAL, INTENT(IN) :: x(n), xp(n), c(n)
  END FUNCTION stopcond

END INTERFACE


```

```
!IF(size(A,2) /= size(c)) STOP "Algorithm ERROR: Wrong size for input argument"

n = size(A,2)
xp = one/n

!IF( ANY(GEMV(A,xp) >= ukind(1.Q-10)) ) STOP "Center of Simplex doesn't lie in Null space
     of the Input Matrix:"

iter = 1
DO

x = Optimize(xp)
IF(stopcond(n, x,xp, iter, c)) EXIT

!IF( ANY(ISNAN(x))) THEN
!PRINT*,  "WARNING: NAN occurred in the Solution"
!x= xp
!EXIT
!END IF

xp = x
iter = iter + 1

END DO

xcan = x

CONTAINS

PURE FUNCTION Optimize(xp) RESULT(x)

UREAL, INTENT(IN) :: xp(:)
UREAL             :: x(size(xp))

UREAL :: Ad(size(A,1),size(xp)), B(size(A,1)+1,size(xp)), &
            v(size(A,1)+1), cp(size(c)), cunit(size(c)), alpha
UREAL ::  e(size(xp)) = one
UREAL :: x0(size(xp)) = e/n


Ad = COLMULT(xp,A)
B = Ad .VAUG. e
v = PDSOL(GEMM(B,TRANS(B)), GEMV(B,xp*c))

cp = xp*c - GEMV(TRANS(B),v)
cunit = cp/ENORM(cp)

alpha = ratiotest(n, cunit)

x = x0 - alpha*cunit
x = xp*x/DOT(xp,x)

END FUNCTION Optimize

END FUNCTION ProjectiveScale
```

ProjScalingAlgo

2 Ratio Test

```
UREAL PURE FUNCTION potentialratio(n, cunit) RESULT(alpha)

INTEGER, INTENT(IN) :: n
UREAL, INTENT(IN) :: cunit(n)


alpha = 1/(4*NSQRT(ukind(n)*(ukind(n)-1)))

END FUNCTION potentialratio
```

PotentialRatio

```
UREAL PURE FUNCTION minratio(n, cunit) RESULT(alpha)

INTEGER, INTENT(IN) :: n
UREAL, INTENT(IN) :: cunit(n)

UREAL, PARAMETER :: beta = ukind(1.Q-1)


alpha = (one-beta)/(n*maxval(cunit))

END FUNCTION minratio
```

MinimumRatio

```
UREAL PURE FUNCTION zeroratio(n, cunit) RESULT(alpha)

INTEGER, INTENT(IN) :: n
UREAL, INTENT(IN) :: cunit(n)

UREAL, PARAMETER :: beta = ukind(1.Q-1)
UREAL :: a
INTEGER :: idx


alpha = one/(n*cunit(n-1))
DO idx=1,n
  IF(cunit(idx)<0 .OR. idx == n-1) CYCLE
  a = (ukind(1.Q0)-beta)/(n*cunit(idx))
  IF(alpha>a) alpha = a
END DO

END FUNCTION zeroratio
```

ZeroRatio

3 Stopping Criteria

```
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

```
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

```
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

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

```
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

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

Introducing an artificial variable to create an interior starting point,

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

This gives us an LPP in Equality form,

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

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

Projectively Transforming to Canonical form,

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

```
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:

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

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

Let

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

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.

```
SUBROUTINE LeastNegative

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

INTEGER :: i, j


CALL ProbGen(Aln)
CALL ProbGen(bln)
CALL ProbGen(cln)

e=one

y0 = PDSOL(GEMM(Aln,TRANS(Aln)), bln)

x0 = GEMV(TRANS(Aln), y0)

IF(ANY(x0 < 0)) THEN

eps = -2*minval(x0)

Aaug = Aln .HAUG. -GEMV(Aln,e)

baug = bln

x0aug = (x0 + eps) .VAUG. eps

caug = 0
caug(column+1) = 1

CALL ProjectiveTransform(Aaug,baug,caug, Aaugcan, caugcan, x0aug)

xaugcan = ProjectiveScale(Aaugcan, caugcan, potentialstop, potentialratio)

xaug = InvProjTransform(xaugcan, x0aug)

x0 = xaug(:column)

END IF

IF(xaug(column+1) < 1.0Q-30) THEN

CALL ProjectiveTransform(Aln,bln,cln, Acan,ccan, x0)

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

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

xln = InvProjTransform(xcan, x0)

ELSE

eps = xaug(column +1)

END IF

END SUBROUTINE LeastNegative
```

LeastNegative

5 Modules

5.1 LPPGeneration

```
#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

```
#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
```

```
  CALL SYSTEM_CLOCK(count)
  seed = MOD(count,10000)
END IF

DO i = 1, size(M,1)
  DO j = 1, size(M,2)
    M(i,j) = ProbDist(seed)
  END DO
END DO

END SUBROUTINE ProbMatGen

END MODULE ProbabilityGeneration
#endif
```

ProbabilityGeneration

5.3 ProbabilityDistributions

```
#include "Preprocessor.F90"

#ifdef ProbDist
MODULE ProbabilityDistribution
IMPLICIT NONE

PRIVATE
PUBLIC :: ProbDist

CONTAINS

FUNCTION Uniform(seed) RESULT(u)
! Uniform Distribution scalar u from integer seed

INTEGER :: seed
INTEGER, PARAMETER :: IA=16807,IM=2147483647,IQ=127773,IR=2836
UREAL, SAVE :: am
INTEGER, SAVE :: ix=-1,iy=-1,k

UREAL :: u


! Initialise
IF (seed <= 0 .OR. iy < 0) THEN
am=nearest(1.0,-1.0)/IM
iy=ior(ieor(888889999,abs(seed)),1)
ix=ieor(777755555,abs(seed))
seed=abs(seed)+1
END IF

!!! Marsaglia shift sequence with period 2^32 -1
ix=ieor(ix,ishft(ix,13))
ix=ieor(ix,ishft(ix,-17))
ix=ieor(ix,ishft(ix,5))

! Park-Miller sequence by SchrageâĂŹs method with period2^31âĹŠ2
k=iy/IQ
iy=IA*(iy-k*IQ)-IR*k
if (iy < 0) iy=iy+IM

! Combine two Generators
u=am*ior(iand(IM,ieor(ix,iy)),1)

#ifdef lowerbound
#ifdef upperbound
u = (upperbound-lowerbound)*u + lowerbound
#endif
```

```
#endif

END FUNCTION Uniform

FUNCTION Normal(seed) RESULT(n)
! Normal Distribution scalar n from integer seed

INTEGER :: seed
UREAL, PARAMETER :: PI = 4.Q0*DATAN(1.D0)
UREAL :: u1,u2

UREAL :: n


u1 =  Uniform(seed)
u2 =  Uniform(seed)

n = sqrt(-2*log(u1))*cos(2*PI*u2)

#ifdef mean
#ifdef variance
n = variance*n + mean
#endif
#endif

END FUNCTION Normal

END MODULE ProbabilityDistribution
#endif
```

ProbabilityDistributions

5.4 LPPMethods

```
#include "Preprocessor.F90"

MODULE LinearProblemSolver
USE ProbabilityGeneration, ONLY : ProbGen
USE ProjScalingAlgo, ONLY : ProjectiveScale
USE LPTools, ONLY : StdToCan, ProjectiveTransform, InvProjTransform, Dual, zerostop,
    potentialstop, optimumstop, zeroratio, minratio, potentialratio
USE LALibrary, ONLY : DOT, TRANS, GEMV, ENORM, PDSOL, GEMM
USE AUGOperator, ONLY : OPERATOR(.VAUG.), OPERATOR(.HAUG.)
IMPLICIT NONE

PRIVATE
PUBLIC :: LPType

CONTAINS

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(c)

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

END SUBROUTINE Canonical

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

SUBROUTINE Standard

UREAL :: Astd(row,column), bstd(row), cstd(column), xopt(column), &
              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)), &
               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)
INTEGER :: i, j


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

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

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

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

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

xopt1 = InvProjTransform(xcan, a0)

xopt = xopt1(:column)

END SUBROUTINE Standard

SUBROUTINE LeastNegative

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

INTEGER :: i, j


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

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

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

```
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

```
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
```

```
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
```

```
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
```

```
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
```

```
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)
```

```
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
```

```
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)
```

```
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

```
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
```

```
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(:)
```

```
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
```

```
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