Applications of Linear Types

UNIVERSITY OF CAMBRIDGE

Dhruv Makwana
Trinity

What are linear types?

$$\Gamma_{\text{in}} \vdash e; \Gamma_{\text{out}}$$

$$\frac{}{\Gamma, x : t \vdash x : t; \Gamma}\ \textsc{Var}$$

$$\frac{\Gamma_1 \vdash e_1 : t_1; \Gamma_2 \qquad \Gamma_2 \vdash e_2 : t_2; \Gamma_3}{\Gamma_1 \vdash (e_1, e_2) : t_1 \otimes t_2; \Gamma_3}\ \textsc{Pair}$$

~~What are linear types?~~

(honestly, just an implementation detail)

Why should I care?

Why Should I care?

```
C = a*b+C;
```

- easy if a,b,c are 'values'
- difficult if it is a matrix expression

```
Form  C := alpha*A*B + beta*C.

 DO 90 J = 1,N
     IF (BETA.EQ.ZERO) THEN
         DO 50 I = 1,M
             C(I,J) = ZERO
         CONTINUE
     ELSE IF (BETA.NE.ONE) THEN
         DO 60 I = 1,M
             C(I,J) = BETA*C(I,J)
         CONTINUE
     END IF
     DO 80 L = 1,K
         IF (B(L,J).NE.ZERO) THEN
             TEMP = ALPHA*B(L,J)
             DO 70 I = 1,M
                 C(I,J) = C(I,J) + TEMP*A(I,L)
             CONTINUE
         END IF
     CONTINUE
 CONTINUE
```

BLAS/
LAPACK

Why Should I care?

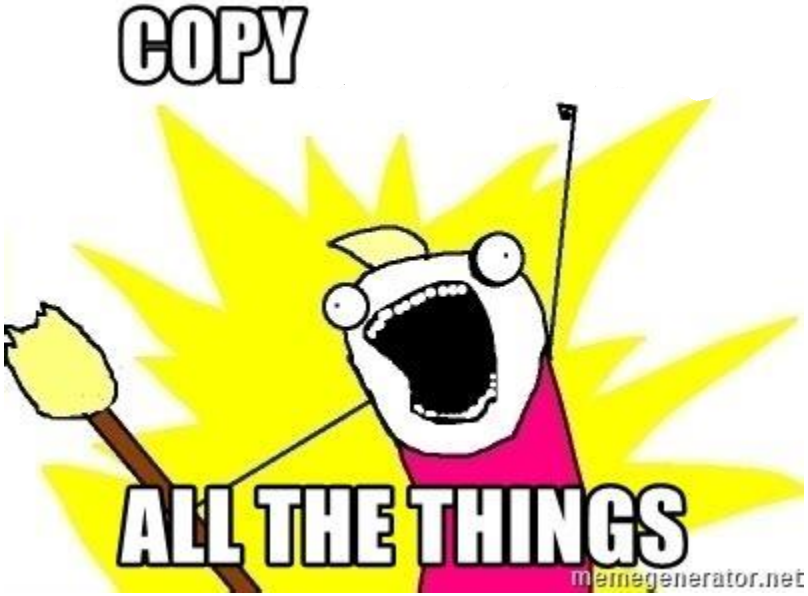
```
C = a*b+C;
```

- easy if a,b,c are 'values'
- ~~difficult if it is a matrix expression~~

Numpy

Julia

Owl



Why Should I care?

```
C = a*b+C;
```

- easy if a,b,c are 'values'
- ~~difficult~~ if it is a matrix expression:
  - a,b may alias each other
  - c must NOT alias a,b
  - assuming no memory overlaps

$$\mu' = \mu + \Sigma H^T (R + H\Sigma H^T)^{-1} (H\mu - data)$$

$$\Sigma' = \Sigma (\mathbb{I} - H^T (R + H\Sigma H^T)^{-1} H\Sigma$$

```
subroutine kalman(mu, Sigma, H, INFO, R, Sigma_2, data, mu_2, k, n)
implicit none

integer, intent(in)    :: k, n
real*8,  intent(in)    :: Sigma(n,n), H(k,n), mu(n)
real*8,  intent(inout) :: data(k)      ! data, H*mu - data , (H*Sigma*H^T + R)^-1*(H*mu - data)
real*8,  intent(inout) :: R(k, k)      ! R, H*Sigma*H^T + R
integer, intent(out)   :: INFO         ! INFO
real*8,  intent(out)   :: Sigma_2(n,n) ! H^T*(H*Sigma*H^T + R)^-1*H, Sigma, Sigma*(I - H^T*(H*Sigma*H^T + R)^-1*H*Sigma)
real*8,  intent(out)   :: mu_2(n)      ! mu, Sigma*H^T*(H*Sigma*H^T + R)^-1*(H*mu - data) + mu
real*8                 :: H_2(k,n)     ! H * Sigma , H , (H*Sigma*H^T + R)^-1*H
real*8                 :: chol_R(k,k)  ! R, U where (H*Sigma*H^T + R)=U^T*U
real*8                 :: H_data(n)    ! H^T*(H*Sigma*H^T + R)^-1*(H*mu - data)
real*8                 :: N_N_tmp(n,n) ! H^T*(H*Sigma*H^T + R)^-1*H*Sigma

call dsymm('R', 'U', k, n, 1, Sigma, n, H, n, 0, H_2, n)           ! H_2     := 1. * H   * Sigma +  0. * H_2
call dgemm('N', 'T', k, k, n, 1, H_2, n, H, n, 1, R, k)            ! R       := 1. * H_2 * H     +  1. * R
call dgemm('N', 'N', k, 1, n, 1, H, n, mu, 1, -1, data, 1)         ! data    := 1. * H   * mu    + -1. * data
call dcopy(k*n, H, 1, H_2, 1)                                      ! H_2     := H
call dcopy(k*k, R, 1, chol_R, 1)                                   ! chol_R  := R
call dposv('U', k, n, chol_R, k, H_2, n, INFO)                     ! chol_R  := U where R = U^T * U
                                                                   ! H_2     := R^-1 * H_2
call dpotrs('U', k, 1, chol_R, k, data, 1, INFO)                   ! data    := R^-1 * data
call dgemm('T', 'N', n, n, k, 1, H, n, H_2, n, 0, Sigma_2, n)      ! N_N_tmp := 1. * H^T * H_2    + 0. * N_N_tmp
call dgemm('T', 'N', n, 1, k, 1, H, n, data, 1, 0, H_data, 1)      ! H_data  := 1. * H^T * data   + 0. * H_data
call dcopy(n, mu, 1, mu_2, 1)                                      ! mu_2    := mu
call dsymm('L', 'U', n, 1, 1, Sigma, n, H_data, 1, 1, mu_2, 1)     ! mu_2    := 1. * Sigma * H_data  + 1. * mu_2
call dsymm('R', 'U', n, n, 1, Sigma, n, Sigma_2, n, 0, N_N_tmp, n) ! N_N_tmp := 1. * N_N_tmp * Sigma   + 0. * N_N_tmp
call dcopy(n**2, Sigma, 1, Sigma_2, 1)                             ! Sigma_2 := Sigma
call dsymm('L', 'U', n, n, -1, Sigma, n, N_N_tmp, n, 1, Sigma_2, n) ! Sigma_2 := -1 * Sigma * N_N_tmp + 1. * Sigma_2

RETURN
END
```

$$\mu' = \mu + \Sigma H^T (R + H\Sigma H^T)^{-1} (H\mu - data)$$

$$\Sigma' = \Sigma (\mathbb{I} - H^T (R + H\Sigma H^T)^{-1} H\Sigma$$

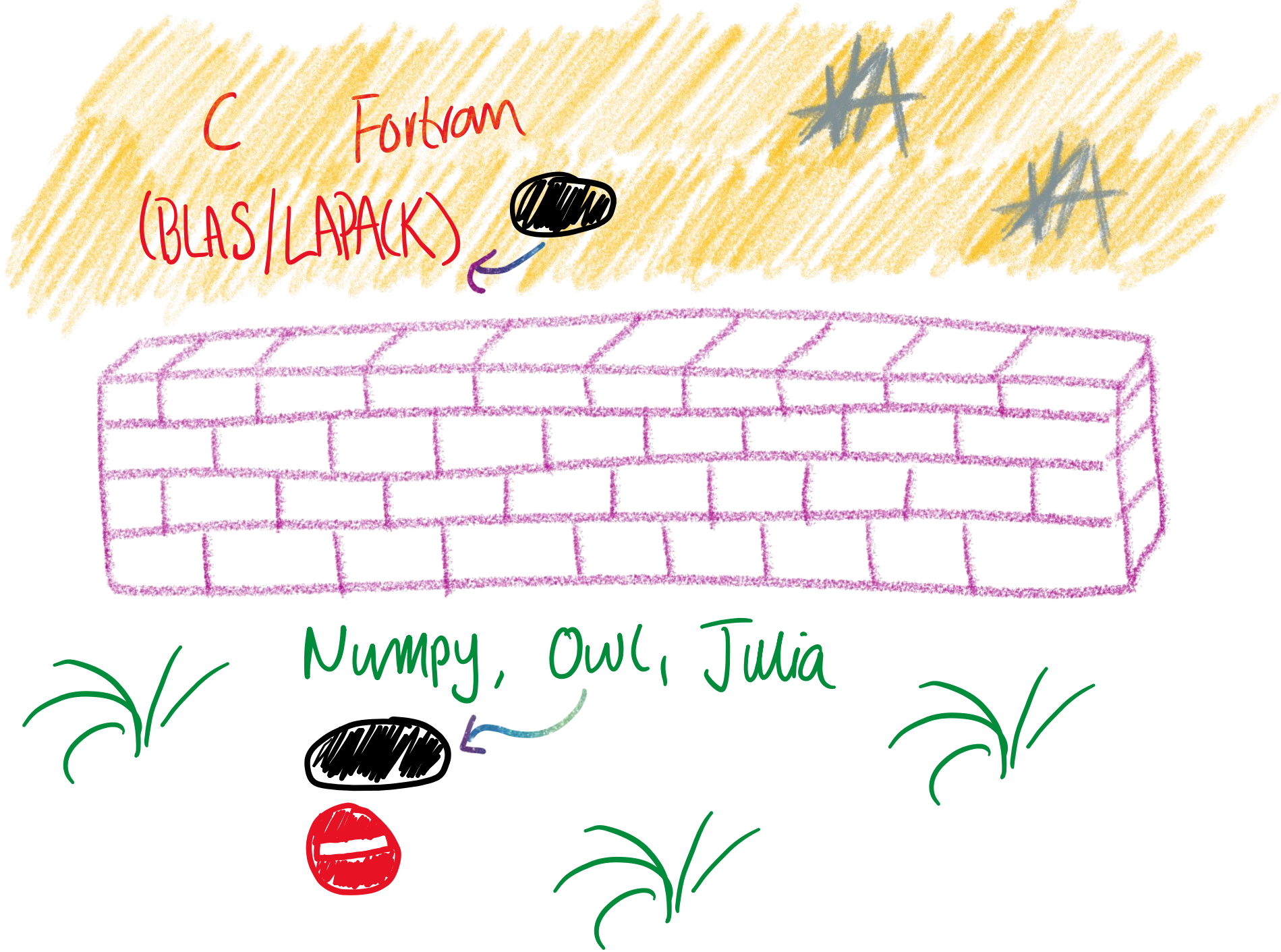
```
subroutine kalman(mu, Sigma, H, INFO, R, Sigma_2, data, mu_2, k, n)
implicit none

integer, intent(in)    :: k, n
real*8,  intent(in)    :: Sigma(n,n), H(k,n), mu(n)
real*8,  intent(inout) :: data(k)       ! data, H*mu - data , (H*Sigma*H^T + R)^-1*(H*mu - data)
real*8,  intent(inout) :: R(k, k)       ! R, H*Sigma*H^T + R
integer, intent(out)   :: INFO          ! INFO
real*8,  intent(out)   :: Sigma_2(n,n) ! H^T*(H*Sigma*H^T + R)^-1*H, Sigma, Sigma*(I - H^T*(H*Sigma*H^T + R)^-1*H*Sigma)
real*8,  intent(out)   :: mu_2(n)       ! mu, Sigma*H^T*(H*Sigma*H^T + R)^-1*(H*mu - data) + mu
real*8                 :: H_2(k,n)      ! H * Sigma , H , (H*Sigma*H^T + R)^-1*H
real*8                 :: chol_R(k,k)   ! R, U where (H*Sigma*H^T + R)=U^T*U
real*8                 :: H_data(n)     ! H^T*(H*Sigma*H^T + R)^-1*(H*mu - data)
real*8                 :: N_N_tmp(n,n) ! H^T*(H*Sigma*H^T + R)^-1*H*Sigma

call dsymm('R', 'U', k, n, 1, Sigma, n, H, n, 0, H_2, n)          ! H_2     := 1. * H   * Sigma +  0. * H_2
call dgemm('N', 'T', k, k, n, 1, H_2, n, H, n, 1, R, k)           ! R       := 1. * H_2 * H     +  1. * R
call dgemm('N', 'N', k, 1, n, 1, H, n, mu, 1, -1, data, 1)        ! data    := 1. * H   * mu    + -1. * data
call dcopy(k*n, H, 1, H_2, 1)                                     ! H_2     := H
call dcopy(k*k, R, 1, chol_R, 1)                                  ! chol_R  := R
call dposv('U', k, n, chol_R, k, H_2, n, INFO)                    ! chol_R  := U where R = U^T * U
                                                                  ! H_2     := R^-1 * H_2
call dpotrs('U', k, 1, chol_R, k, data, 1, INFO)                  ! data    := R^-1 * data
call dgemm('T', 'N', n, n, k, 1, H, n, H_2, n, 0, Sigma_2, n)     ! N_N_tmp := 1. * H^T * H_2    + 0. * N_N_tmp
call dgemm('T', 'N', n, 1, k, 1, H, n, data, 1, 0, H_data, 1)     ! H_data  := 1. * H^T * data   + 0. * H_data
call dcopy(n, mu, 1, mu_2, 1)                                     ! mu_2    := mu
call dsymm('L', 'U', n, 1, 1, Sigma, n, H_data, 1, 1, mu_2, 1)    ! mu_2    := 1. * Sigma * H_data  + 1. * mu_2
call dsymm('R', 'U', n, n, 1, Sigma, n, Sigma_2, n, 0, N_N_tmp, n) ! N_N_tmp := 1. * N_N_tmp * Sigma   + 0. * N_N_tmp
call dcopy(n**2, Sigma, 1, Sigma_2, 1)                            ! Sigma_2 := Sigma
call dsymm('L', 'U', n, n, -1, Sigma, n, N_N_tmp, n, 1, Sigma_2, n) ! Sigma_2 := -1 * Sigma * N_N_tmp + 1. * Sigma_2

RETURN
END
```

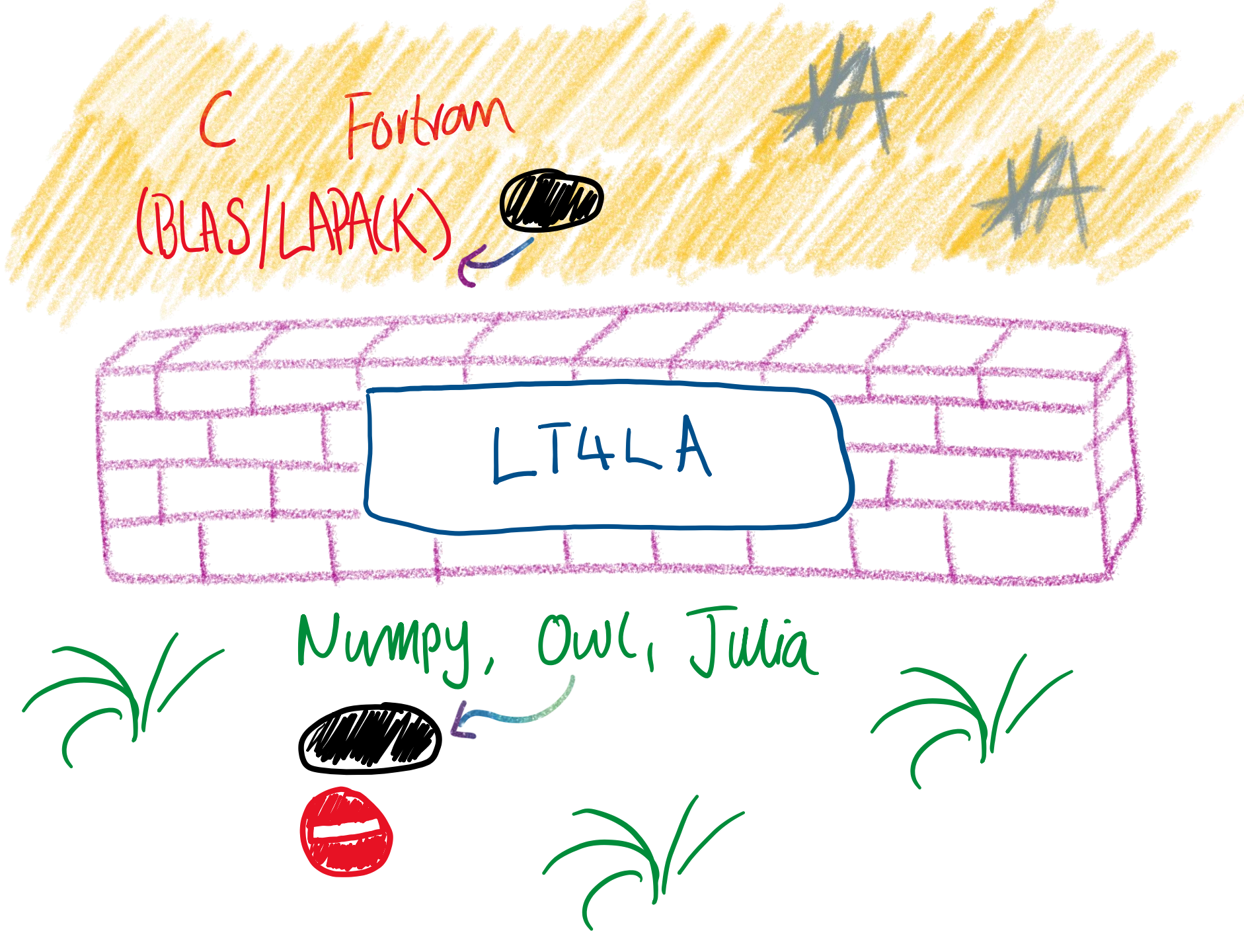
```
! H_2       := 1. * H   * Sigma +  0. * H_2
! R         := 1. * H_2 * H     +  1. * R
! data      := 1. * H   * mu    + -1. * data
! H_2       := H
! chol_R    := R
! chol_R    := U where R = U^T * U
! H_2       := R^-1 * H_2
! data      := R^-1 * data
! N_N_tmp   := 1. * H^T * H_2     + 0. * N_N_tmp
! H_data    := 1. * H^T * data    + 0. * H_data
! mu_2      := mu
! mu_2      := 1. * Sigma * H_data  + 1. * mu_2
! N_N_tmp   := 1. * N_N_tmp * Sigma   + 0. * N_N_tmp
! Sigma_2   := Sigma
! Sigma_2   := -1 * Sigma * N_N_tmp + 1. * Sigma_2
```

$$\mu' = \mu + \Sigma H^T (R + H\Sigma H^T)^{-1} (H\mu - data)$$
$$\Sigma' = \Sigma (\mathbb{I} - H^T (R + H\Sigma H^T)^{-1} H\Sigma$$

```
subroutine kalman(mu, Sigma, H, INFO, R, Sigma_2, data, mu_2, k, n)
implicit none

integer, intent(in)    :: k, n
real*8,  intent(in)    :: Sigma(n,n), H(k,n), mu(n)
real*8,  intent(inout) :: data(k)       ! data, H*mu - data , (H*Sigma*H^T + R)^-1*(H*mu - data)
real*8,  intent(inout) :: R(k, k)       ! R, H*Sigma*H^T + R
integer, intent(out)   :: INFO          ! INFO
real*8,  intent(out)   :: Sigma_2(n,n)  ! H^T*(H*Sigma*H^T + R)^-1*H, Sigma, Sigma*(I - H^T*(H*Sigma*H^T + R)^-1*H*Sigma)
real*8,  intent(out)   :: mu_2(n)       ! mu, Sigma*H^T*(H*Sigma*H^T + R)^-1*(H*mu - data) + mu
real*8                 :: H_2(k,n)      ! H * Sigma , H , (H*Sigma*H^T + R)^-1*H
real*8                 :: chol_R(k,k)   ! R, U where (H*Sigma*H^T + R)=U^T*U
real*8                 :: H_data(n)     ! H^T*(H*Sigma*H^T + R)^-1*(H*mu - data)
real*8                 :: N_N_tmp(n,n)  ! H^T*(H*Sigma*H^T + R)^-1*H*Sigma

call dsymm('R', 'U', k, n, 1, Sigma, n, H, n, 0, H_2, n)          ! H_2     := 1. * H   * Sigma + 0. * H_2
call dgemm('N', 'T', k, k, n, 1, H_2, n, H, n, 1, R, k)           ! R       := 1. * H_2 * H     + 1. * R
call dgemm('N', 'N', k, 1, n, 1, H, n, mu, 1, -1, data, 1)        ! data    := 1. * H   * mu    + -1. * data
call dcopy(k*n, H, 1, H_2, 1)                                     ! H_2     := H
call dcopy(k*k, R, 1, chol_R, 1)                                  ! chol_R  := R
call dposv('U', k, n, chol_R, k, H_2, n, INFO)                    ! chol_R  := U where R = U^T * U
                                                                  ! H_2     := R^-1 * H_2
call dpotrs('U', k, 1, chol_R, k, data, 1, INFO)                  ! data    := R^-1 * data
call dgemm('T', 'N', n, n, k, 1, H, n, H_2, n, 0, Sigma_2, n)     ! N_N_tmp := 1. * H^T * H_2    + 0. * N_N_tmp
call dgemm('T', 'N', n, 1, k, 1, H, n, data, 1, 0, H_data, 1)     ! H_data  := 1. * H^T * data   + 0. * H_data
call dcopy(n, mu, 1, mu_2, 1)                                     ! mu_2    := mu
call dsymm('L', 'U', n, 1, 1, Sigma, n, H_data, 1, 1, mu_2, 1)    ! mu_2    := 1. * Sigma * H_data + 1. * mu_2
call dsymm('R', 'U', n, n, 1, Sigma, n, Sigma_2, n, 0, N_N_tmp, n)! N_N_tmp := 1. * N_N_tmp * Sigma + 0. * N_N_tmp
call dcopy(n**2, Sigma, 1, Sigma_2, 1)                            ! Sigma_2 := Sigma
call dsymm('L', 'U', n, n, -1, Sigma, n, N_N_tmp, n, 1, Sigma_2, n) ! Sigma_2 := -1 * Sigma * N_N_tmp + 1. * Sigma_2

RETURN
END
```

```
let owl_kalman ~sigma ~h ~mu ~r ~data =
  let open Owl.Mat in
  let ( * ) = dot in
  let h' = transpose h in
  let sigma_h' = sigma * h' in
  let x = sigma_h' * (inv @@ r + h * sigma_h') in
  let new_mu = mu + x * (h * mu - data) in
  let new_sigma = sigma - x * h * sigma in
  ((sigma, (h, (mu, (r, data)))), (new_mu, new_sigma))
;;
```


C
Fortran
(BLAS/LAPACK)
Numpy, Owl, Julia


C
Fortran
(BLAS/LAPACK)
LT4LA
Numpy, Owl, Julia

usable library
for linear algebra

LT4LA

and matrix expressions
which is safe AND explicit

$$
\begin{aligned}
f ::= \\
&\mid fc \\
&\mid \mathbf{Z} \\
&\mid \mathbf{S}\ f \\
t ::= \\
&\mid \mathbf{unit} \\
&\mid \mathbf{bool} \\
&\mid \mathbf{int} \\
&\mid \mathbf{elt} \\
&\mid f\ \mathbf{arr} \\
&\mid f\ \mathbf{mat} \\
&\mid \mathbf{!}\ t \\
&\mid \forall\ fc.\ t \\
&\mid t \otimes t' \\
&\mid t \multimap t'
\end{aligned}
$$

```
module Arr =
  Owl.Dense.Ndarray.D

type z = Z
type 'a s = Succ

type 'a arr =
  A of Arr.arr
  [@@unboxed]

type 'a mat =
  M of Arr.arr
  [@@unboxed]

type 'a bang =
  Many of 'a
  [@@unboxed]
```

$$
\begin{aligned}
[\![fc]\!] &= \texttt{'fc} \\
[\![\mathbf{Z}]\!] &= \texttt{z} \\
[\![\mathbf{S}\ f]\!] &= [\![f]\!]\ \texttt{s} \\
[\![\mathbf{unit}]\!] &= \texttt{unit} \\
[\![\mathbf{bool}]\!] &= \texttt{bool} \\
[\![\mathbf{int}]\!] &= \texttt{int} \\
[\![\mathbf{elt}]\!] &= \texttt{float} \\
[\![f\ \mathbf{arr}]\!] &= [\![f]\!]\ \texttt{arr} \\
[\![f\ \mathbf{mat}]\!] &= [\![f]\!]\ \texttt{mat} \\
[\![\mathbf{!}\ t]\!] &= [\![t]\!]\ \texttt{bang} \\
[\![\forall fc.\ t]\!] &= [\![t]\!] \\
[\![t \otimes t']\!] &= [\![t]\!] * [\![t']\!] \\
[\![t \multimap t']\!] &= [\![t]\!] \rightarrow [\![t']\!]
\end{aligned}
$$

$$
\begin{array}{lll}
e & ::= & \\
& \mid & p \\
& \mid & x \\
& \mid & \mathbf{let}\ x = e\ \mathbf{in}\ e' \qquad \mathsf{bind}\ x\ \mathsf{in}\ e' \\
& \mid & () \\
& \mid & \mathbf{let}\ () = e\ \mathbf{in}\ e' \\
& \mid & \mathbf{true} \\
& \mid & \mathbf{false} \\
& \mid & \mathbf{if}\ e\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \\
& \mid & k \\
& \mid & el \\
& \mid & \mathbf{Many}\ e \\
& \mid & \mathbf{let}\ \mathbf{Many}\ x = e\ \mathbf{in}\ e' \\
& \mid & \mathbf{fun}\ fc \to e \\
& \mid & e[f] \\
& \mid & (e, e') \\
& \mid & \mathbf{let}\ (a, b) = e\ \mathbf{in}\ e' \qquad \mathsf{bind}\ a \cup b\ \mathsf{in}\ e' \\
& \mid & \mathbf{fun}\ x : t \to e \qquad \mathsf{bind}\ x\ \mathsf{in}\ e \\
& \mid & e\ e' \\
& \mid & \mathbf{fix}\ (g, x : t, e : t') \qquad \mathsf{bind}\ g \cup x\ \mathsf{in}\ e
\end{array}
$$

```
let rec f (!i : !int) (!n : !int) (!x0 : !elt)
          ('x) (row : 'x arr) : 'x arr * !elt =
    if i = n then
        (row, x0)
    else
        let (row, !x1) = row[i] in
        f (i + 1) n (x0 +. x1) 'x row in
    f
;;
```

```
let rec f i n x0 row =
  if Prim.extract (Prim.eqI i n) then (row, x0)
  else
    let row, x1 = Prim.get row i in
    f (Prim.addI i (Many 1)) n (Prim.addE x0 x1) row
```

```
let row = row[i] := x1 in (* or *) let () = free row in
(* Could not show equality:                                          *)
(*     z arr                                                         *)
(* with                                                              *)
(*     'x arr                                                        *)
(*                                                                   *)
(* Var 'x is universally quantified                                  *)
(* Are you trying to write to/free/unshare an array you don't own? *)
(* In test/examples/sum_array.lt, at line: 7 and column: 19          *)
```

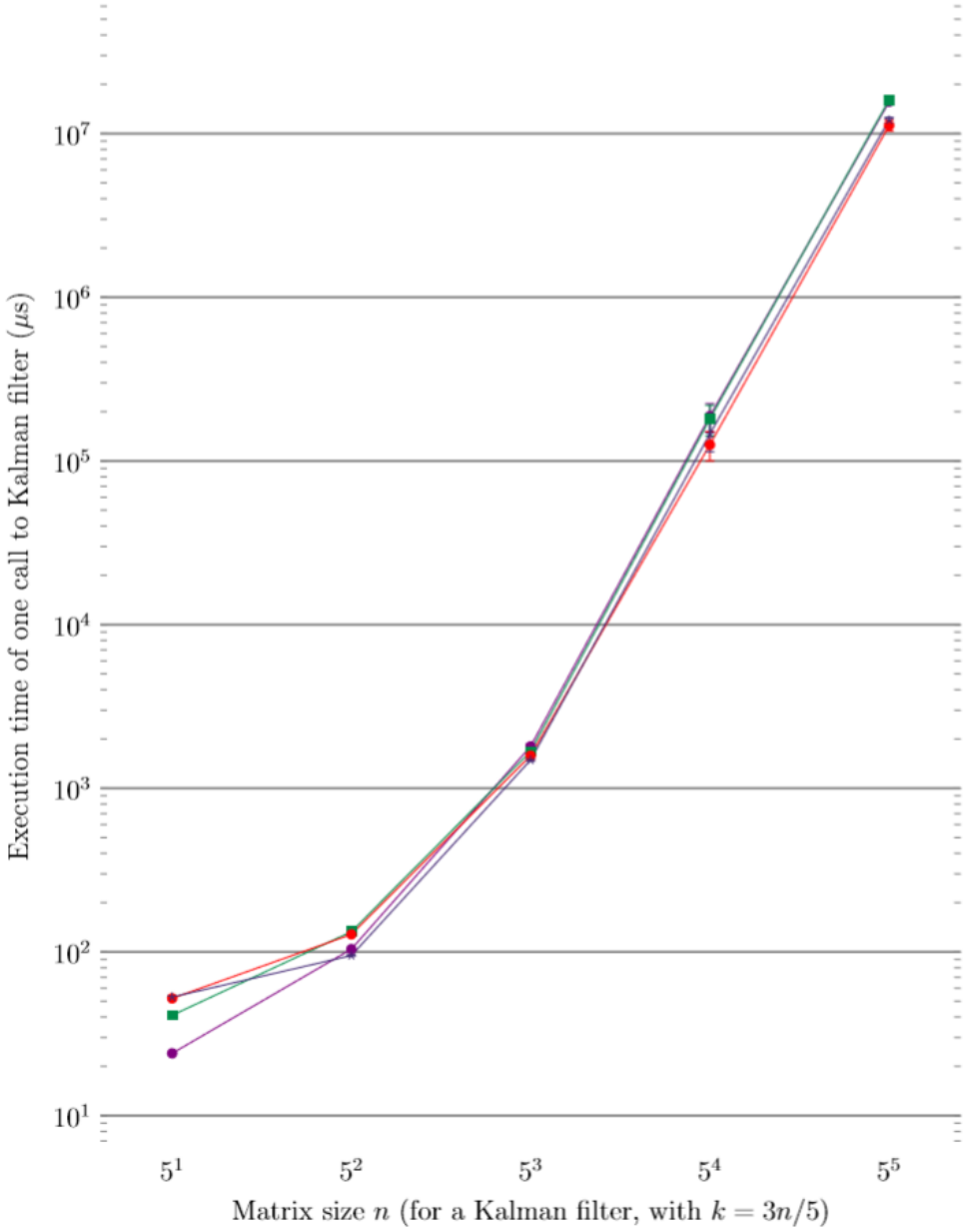
```
(** Arrays *)
val set : z arr -> int bang -> float bang -> z arr
val get : 'a arr -> int bang -> 'a arr * float bang
val share : 'a arr -> 'a s arr * 'a s arr
val unshare : 'a s arr -> 'a s arr -> 'a arr
val free : z arr -> unit
(** Owl *)
val array : int bang -> z arr
val copy : 'a arr -> 'a arr * z arr
(* Level 3 BLAS/LAPACK *)
val gemm : float bang -> ('a mat * bool bang) -> ('b mat * bool bang) ->
           float bang -> z mat -> ('a mat * 'b mat) * z mat
val symm : bool bang -> float bang -> 'a mat -> 'b mat ->
           float bang -> z mat -> ('a mat * 'b mat) * z mat
val posv : z mat -> z mat -> z mat * z mat
val potrs : 'a mat -> z mat -> 'a mat * z mat
```

```
let x <- [| y |] in ..                ==> let (y,x) = copyM_to _ y x in ..
let x <- new [| y |] in ..            ==> let (y,x) = copyM _ y in ..
let x <- [| a*b + c |] in ..          ==> let x <- [| 1.*a*b + 1.*c |] in ..
let x <- [| i*a*b + j*c |] in ..      ==> let ((a,b), c) = (* BLAS *) in ..
let x <- new (m, n) [| a*b |] in .. ==> let c = matrix m n in
                                          let x <- [| 1.*a*b + 0.*c |] in ..
```

```
let !kalman
  ('s) (sigma : 's mat) (* n,n *)
  ('h) (h : 'h mat)     (* k,n *)
  ('m) (mu : 'm mat)    (* n,1 *)
  (r_1 : z mat)         (* k,k *)
  (data_1 : z mat)      (* k,1 *) =
  let (h, (!k, !n)) = sizeM _ h in
  let sigma_h <- new (k, n) [| h * sym (sigma) |] in
  let r_2 <- [| sigma_h * h^T + r_1 |] in
  let data_2 <- [| h * mu - data_1 |] in
  let (h, new_h) = copyM_to _ h sigma_h in
  let new_r <- new [| r_2 |] in
  let (chol_r, sol_h) = posv new_r new_h in
  let (chol_r, sol_data) = potrs _ chol_r data_2 in
  let () = freeM (* k,k *) chol_r in
  let h_sol_h <- new (n, n) [| h^T * sol_h |] in
  let () = freeM (* k,n *) sol_h in
  let h_sol_data <- new (n, 1) [| h^T * sol_data |] in
  let mu_copy <- new [| mu |] in
  let new_mu <- [| sym (sigma) * h_sol_data + mu_copy |] in
  let () = freeM (* n,1 *) h_sol_data in
  let h_sol_h_sigma <- new (n,n) [| h_sol_h * sym(sigma) |] in
  let (sigma, sigma_copy) = copyM_to _ sigma h_sol_h in
  let new_sigma <- [| sigma_copy - sym (sigma) * h_sol_h_sigma |] in
  let () = freeM (* n,n *) h_sol_h_sigma in
  ((sigma, (h, (mu, (r_2, sol_data)))), (new_mu, new_sigma)) in
kalman
;;
```

```
let kalman sigma h mu r_1 data_1 =
  let h, _p_k_n_p_ = Prim.size_mat h in
  let k, n = _p_k_n_p_ in
  let sigma_h = Prim.matrix k n in
  let (sigma, h), sigma_h =
    Prim.symm (Many true) (Many 1.) sigma h (Many 0.) sigma_h
  in
  let (sigma_h, h), r_2 =
    Prim.gemm (Many 1.) (sigma_h, Many false) (h, Many true) (Many 1.) r_1
  in
  let (h, mu), data_2 =
    Prim.gemm (Many 1.) (h, Many false) (mu, Many false) (Many (-1.))
      data_1
  in
  let h, new_h = Prim.copy_mat_to h sigma_h in
  let r_2, new_r = Prim.copy_mat r_2 in
  let chol_r, sol_h = Prim.posv new_r new_h in
  let chol_r, sol_data = Prim.potrs chol_r data_2 in
  let () = Prim.free_mat chol_r in
  let h_sol_h = Prim.matrix n n in
  let (h, sol_h), h_sol_h =
    Prim.gemm (Many 1.) (h, Many true) (sol_h, Many false) (Many 0.)
      h_sol_h
  in
  let () = Prim.free_mat sol_h in
  let h_sol_data = Prim.matrix n (Many 1) in
  let (h, sol_data), h_sol_data =
    Prim.gemm (Many 1.) (h, Many true) (sol_data, Many false) (Many 0.)
      h_sol_data
  in
  let mu, mu_copy = Prim.copy_mat mu in
  let (sigma, h_sol_data), new_mu =
    Prim.symm (Many false) (Many 1.) sigma h_sol_data (Many 1.) mu_copy
  in
  let () = Prim.free_mat h_sol_data in
  let h_sol_h_sigma = Prim.matrix n n in
  let (sigma, h_sol_h), h_sol_h_sigma =
    Prim.symm (Many true) (Many 1.) sigma h_sol_h (Many 0.) h_sol_h_sigma
  in
  let sigma, sigma_copy = Prim.copy_mat_to sigma h_sol_h in
  let (sigma, h_sol_h_sigma), new_sigma =
    Prim.symm (Many false) (Many (-1.)) sigma h_sol_h_sigma (Many 1.)
      sigma_copy
  in
  let () = Prim.free_mat h_sol_h_sigma in
  ((sigma, (h, (mu, (r_2, sol_data)))), (new_mu, new_sigma))
```

```
static void kalman( const int n,                const int k,                  const double *sigma, /* n,n */
                    const double *h, /* k,n */ const double *mu, /* n,1 */ double *r,           /* k,k */
                    double *data,    /* k,1 */ double **ret_mu,  /* k,1 */ double **ret_sigma   /* n,n */ ) {
        double* k_by_n = (double *) malloc(k * n * sizeof(double));
/*16*/  cblas_dsymm(CblasRowMajor, CblasRight, CblasUpper, k, n, 1., sigma, n, h, n, 0., k_by_n, n);
/*17*/  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, k, k, n, 1., k_by_n, n, h, n, 1., r, k);
/*18*/  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, k, 1, n, 1., h, n, mu, 1, -1., data, 1);
/*19*/  cblas_dcopy(k * n, h, 1, k_by_n, 1);
        double* k_by_k = (double *) malloc(k * k * sizeof(double));
/*20*/  cblas_dcopy(k * k, r, 1, k_by_k, 1);
/*21*/  LAPACKE_dposv(LAPACK_ROW_MAJOR, 'U', k, n, k_by_k, k, k_by_n, n);
/*23*/  LAPACKE_dpotrs(LAPACK_ROW_MAJOR, 'U', k, 1, k_by_k, k, data, 1);
        free(k_by_k);
        double* n_by_n = (double *) malloc(n * n * sizeof(double));
/*24*/  cblas_dgemm(CblasRowMajor, CblasTrans, CblasNoTrans, n, n, k, 1., h, n, k_by_n, n, 0., n_by_n, n);
        free(k_by_n);
        double* n_by_1 = (double *) malloc(n * sizeof(double));
/*25*/  cblas_dgemm(CblasRowMajor, CblasTrans, CblasNoTrans, n, 1, k, 1., h, n, data, 1, 0., n_by_1, 1);
        double* new_mu = (double *) malloc(n * sizeof(double));
/*26*/  cblas_dcopy(n, mu, 1, new_mu, 1);
/*27*/  cblas_dsymm(CblasRowMajor, CblasLeft, CblasUpper, n, 1, 1., sigma, n, n_by_1, 1, 1., new_mu, 1);
        free(n_by_1);
        double* n_by_n2 = (double *) malloc(n * n * sizeof(double));
/*28*/  cblas_dsymm(CblasRowMajor, CblasRight, CblasUpper, n, n, 1., sigma, n, n_by_n, n, 0., n_by_n2, n);
/*29*/  cblas_dcopy(n*n, sigma, 1, n_by_n, 1);
/*30*/  cblas_dsymm(CblasRowMajor, CblasLeft, CblasUpper, n, n, -1., sigma, n, n_by_n2, n, 1., n_by_n, n);
        free(n_by_n2);
        *ret_sigma = n_by_n;
        *ret_mu = new_mu; }
```


Execution time of one call to Kalman filter (μs)
10^7
10^6
10^5
10^4
10^3
10^2
10^1
5^1
5^2
5^3
5^4
5^5
Matrix size n (for a Kalman filter, with $k = 3n/5$)
CBLAS
LT4LA
Chol
Owl

Other Things

- value restriction
- flaws with matrix expression compilers
- memory is just one performance factor
- future work: size types, dependent types,
  staging, as an IR for matrix. exp. comp.
- what about Rust?

Plugs & Info

ATypical CompSci

github.com/dc-mak

dhruvmakwana.com