Award #: 1835909

CSSI Element: *libkrylov* -a Modular Open-Source Software Library for Extremely Large Eigenvalue and Linear Problems

Ziyue Shen, Naje' George, Samuel Bekoe, Luke Nambi Mohanam, PI: Filipp Furche
Institution: University of California, Irvine – Department of Chemistry

Introduction

Extremely large and dense linear problems(Eg. **Fig. 1**):

- Extremely large dimension n – explicit storage of a coefficient matrix ($\mathbf{A}$) is *impossible*
- Random sparsity – access to matrix-vector products ($\mathbf{AV}$) is *only* possible "on the fly" , using external code engines
- Matrix-vector multiplication is *much* more expensive compared to vector-vector operations.
- Number of desired solutions, $p \ll n$

These problems are at the core of grand challenge applications in science and engineering:

- fluid dynamics[4]
- wave propagation and in elastic stability[5]
- many-body quantum mechanics[6]
- anomalous diffusion modeling.[7]

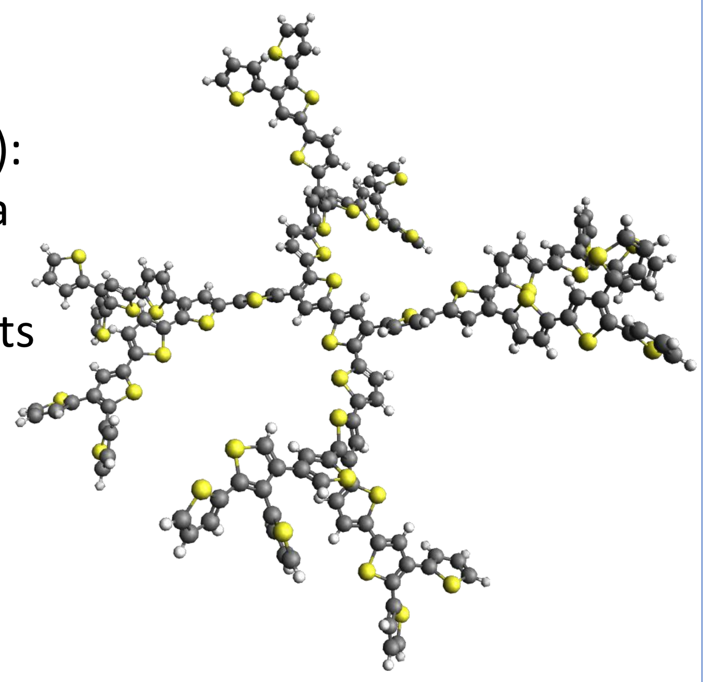
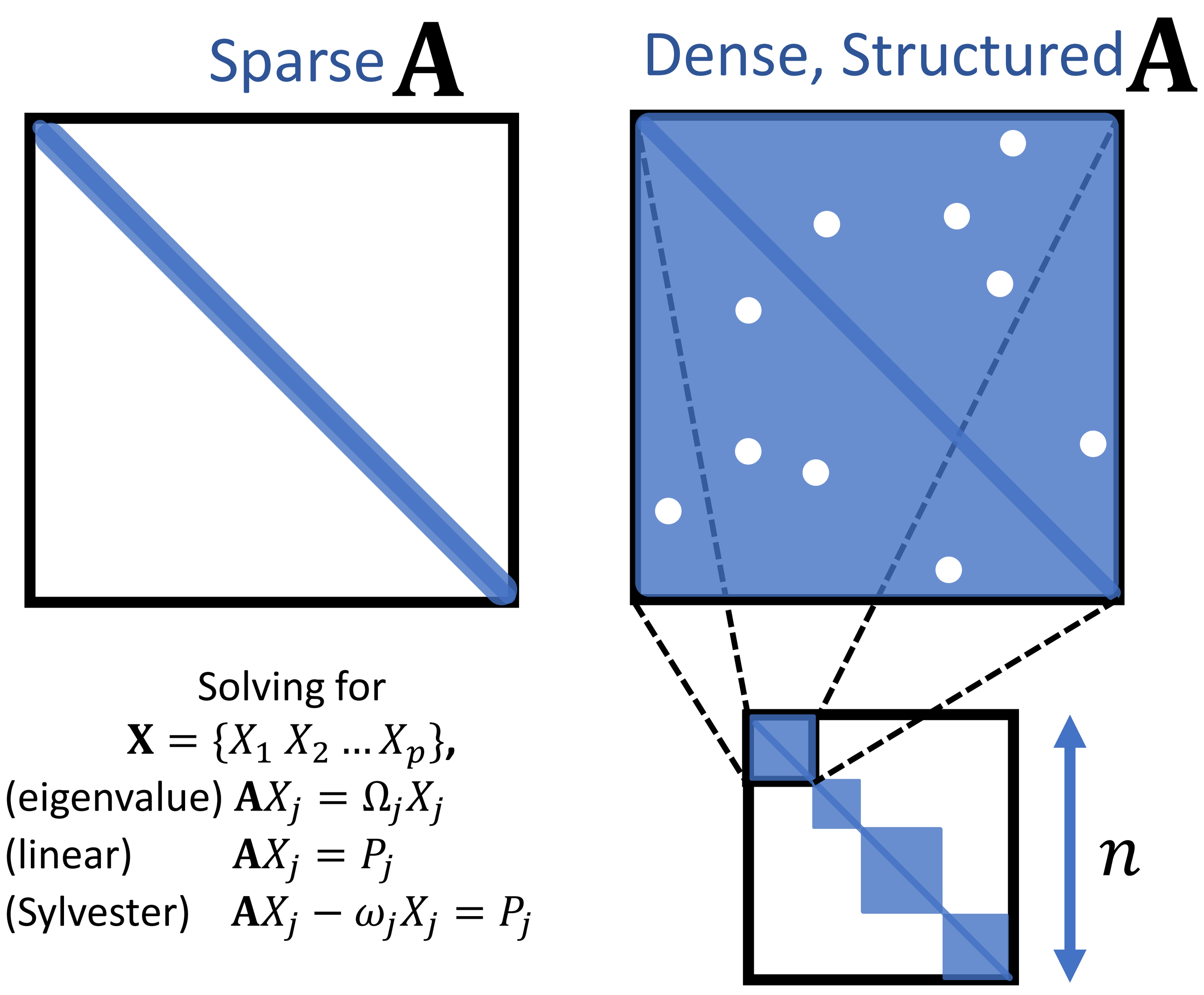


Figure 1. Computation of optical properties of a 42- unit polythiophene dendrimer[1] using TDDFT and def2-TZVPPD basis sets[2,3] corresponds to an eigenvalue problem with n = 15782742, requiring over 14TB storage.



Solving for

$\mathbf{X} = \{X_1\, X_2 \ldots X_p\}$,

(eigenvalue) $\mathbf{A}X_j = \Omega_j X_j$

(linear) $\mathbf{A}X_j = P_j$

(Sylvester) $\mathbf{A}X_j - \omega_j X_j = P_j$

Figure 2. Visualizing Sparsity of Linear Problems

Intended Solver Features

The following features are required for desired applications of *libkrylov*:

F1. Runs on workstations and clusters
F2. Open Source (3-Clause BSD) ✓
F3. Portability ✓
F4. Multi-language Compatibility
F5. Element-type & -precision Agnostic ✓
F6. Minimize costly matrix-vector multiplication[8,9]
F7. Default Non-orthonormal Projection[10-12] ✓
F8. Special Structure Support
F9. Subroutines/Functions as input ✓
F10. Preconditioning Residuals ✓
F11. *A posteriori* error bounds ✓
F12. Dynamic Restarting and Checkpointing[12]
F13. Verbosity Levels

Implementation

The Fortran08 standard was followed[13],

using the GNU fortran compiler, make, autoconf tools and autoconf-archive F3.✓

```
!!Define parameters at compile time
  !! Double precision
  integer, parameter :: kind_float =  &
  & kind_double
```

F5.✓

```
!! Real matrix elements
  type :: base
    real(kind_float) :: element
  end type base
```

```
!!Example User-written module
module user_functions
  !! abstract_type is matrix element
specific
  type, extends(abstract_type) :: user_f
    ⋮
    !!Define arguments for user function
    integer :: input
    real, pointer :: output
    ⋮
  contains
    !!Matching abstract type used to
    !!  define solver
    procedure :: deferred => mvproduct
  end type
end module user_functions
```

```
!! Variable declaration,
!!  normalization routine
  integer, intent(in) :: n1 !!nbasis
  integer, intent(in) :: n2 !!nroots
  type(base), intent(inout) :: vectors(n1,n2)
  integer, intent(inout) :: ierr
  type(base) :: norm_sq_base
  real(kind_float) :: norm_real
  integer:: j !!Do loop dummy variable
```

```
!!Example use of User-written module
  !!Loading modules to call libkrylov
  ⋮
  use libkrylov !!Link .mod and .a
  use user_functions !!Link .o and .mod
  ⋮
  type(user_f) :: functions
  ⋮
  !!passing arguments directly
  !!  to user function
  functions%input  = input
  functions%output => output
  ierr = 0
  !!Call matrix element specific solver
  call libkrylov_solver(functions,ierr)
  ⋮
```

F9.✓

Iterative Algorithm

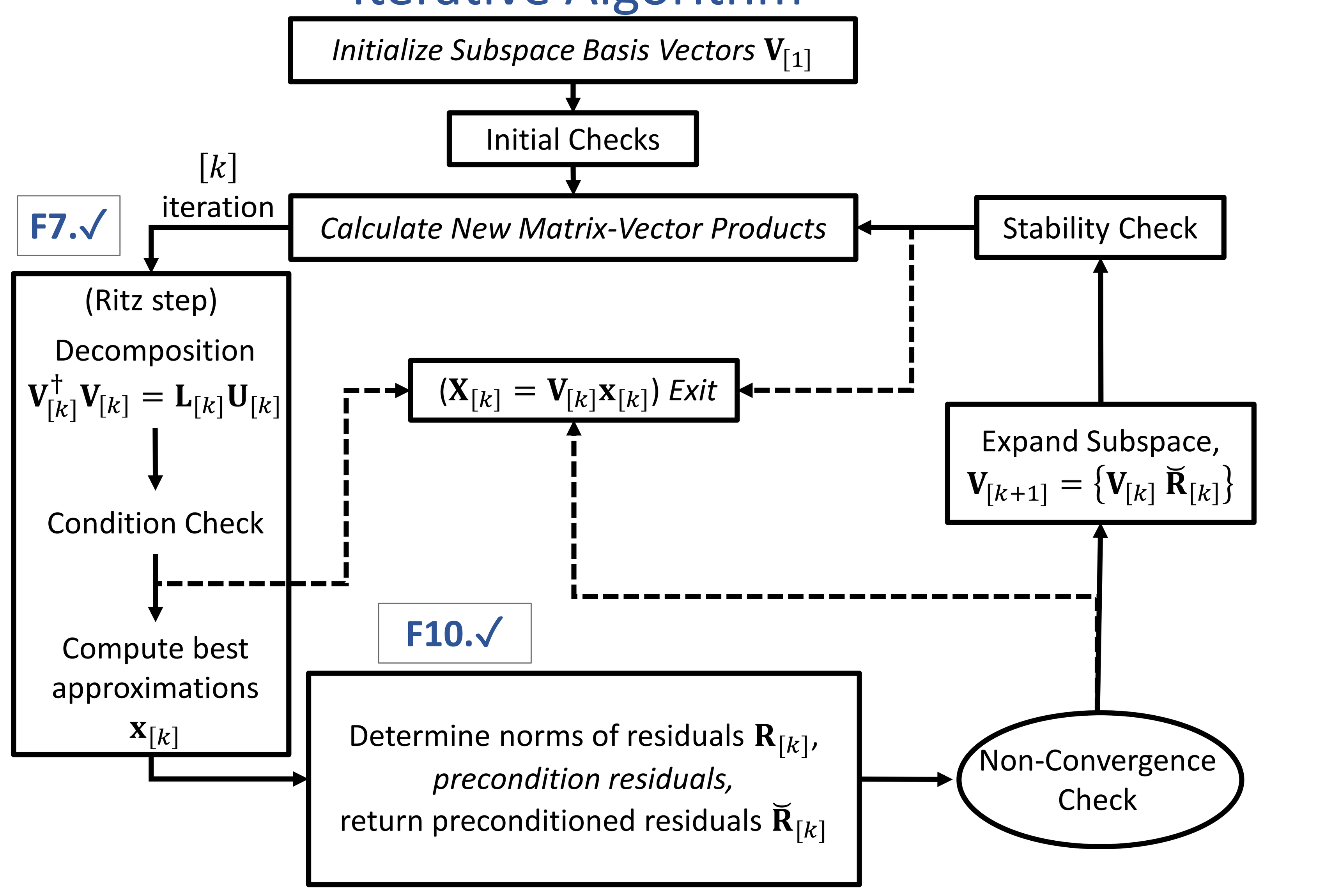


Figure 3. Algorithm as implemented; subroutines in black boxes, functions written by user in *italics;* solid arrows for normal operation, dashed arrows for failing a check.[5]

Convergence

Table 1. $n = 100$, $p = 5$ double precision, real number symmetric eigenvalue problem, Davidson preconditioner.[12] SC is the self consistent solution. The inverse condition number of the Cholesky decomposition is reported. Machine precision $\sim 2.2 \times 10^{-16}$.

k	Ω_3	Inverse condition number
1	4.1971270050481415	
2	3.4720156589122779	2.7989592063603053E-01
3	3.3049070825033655	1.8297041937554334E-01
4	3.2570865837259801	7.1725741466241222E-02
5	3.2462257942912180	1.8917050902269455E-02
6	3.2442049688468311	1.0495496139234563E-02
7	3.2439356309031413	8.2848417107536515E-03
8	3.2439123052360288	6.9345701637029298E-03
9	3.2439104089737194	6.3913876341032015E-03
10	3.2439101752927364	5.9415440595241113E-03
11	3.2439101534708379	5.4766342900749165E-03
12	3.2439101514268698	4.6845733436710786E-03
13	3.2439101512877357	1.4475650472068231E-03
14	3.2439101512831749	9.0346885940684402E-04
15	3.2439101512830475	4.0577818496675229E-04
16	3.2439101512830470	3.9236954031730630E-04
17	3.2439101512830506	2.1144130811136198E-04
SC	3.2439101512830470	

A posteriori Error Bounds

Residuals vectors are determined as follows for the corresponding problem:

(eigenvalue)

$$R_{j[k]} = [(\mathbf{AV}_{[k]})x_{j[k]}] - [\mathbf{V}_{[k]}(x_{j[k]}\Omega_{j[k]})]$$

(linear)

$$R_{j[k]} = [(\mathbf{AV}_{[k]})x_{j[k]}] - P_j$$

(Sylvester)

$$R_{j[k]} = [(\mathbf{AV}_{[k]})x_{j[k]}] - [\mathbf{V}_{[k]}(x_{j[k]}\omega_{j[k]})] - P_j$$

F11.✓ And the error bound is:

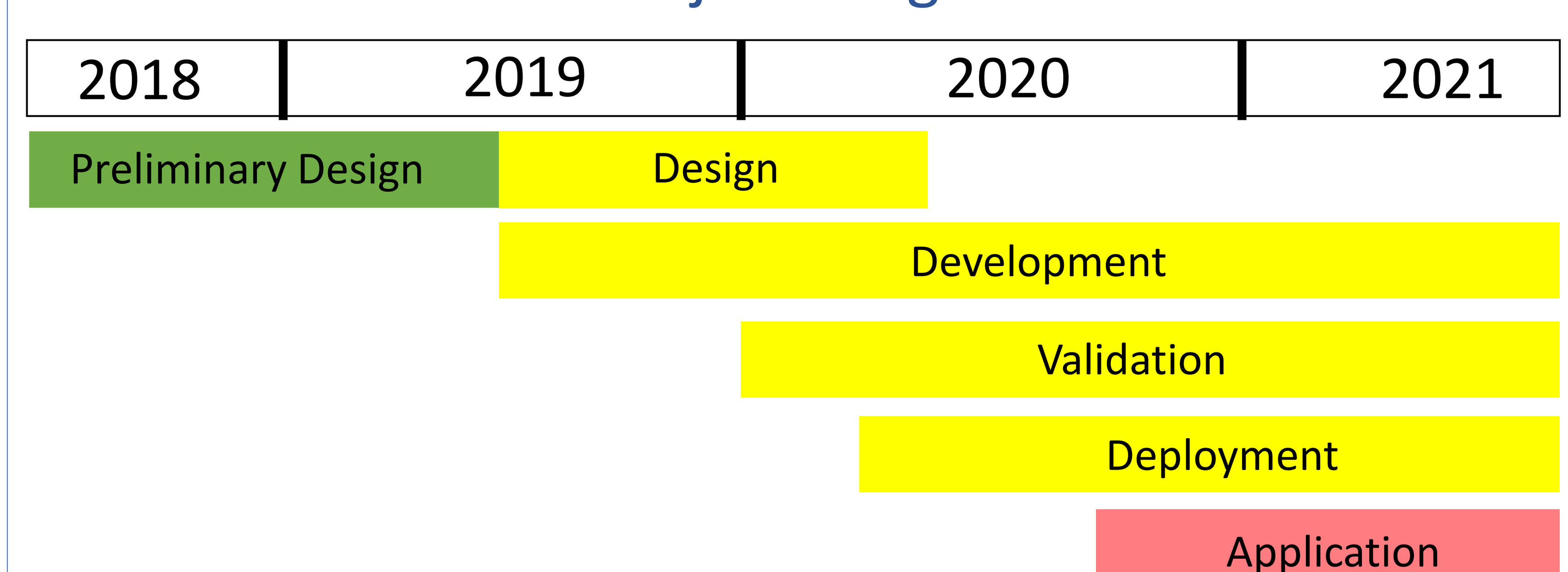
$$\mathrm{error}(\Omega_{j[k]}) = C_\Omega \left(\left\|R_{j[k]}\right\|_2\right)^2$$

$$\mathrm{error}(X_{j[k]}) = C_X \left\|R_{j[k]}\right\|_2$$

Lessons Learned

- Compile-time polymorphism strikes a good balance between generic code and efficiency.
- Hermitian Solver and library built and unit-tested.
- Robust interfacing is enforced by the strictness of Fortran08

Project Progress



References

[1] E. Badaeva, M. R. Harpham, R. Guda, O. Suzer, C.-Q. Ma, P. Bauerle, T. Goodson III, and S. Tretiak, J. Phys. Chem. B 114, 15808–15817, 2010.
[2] F. Weigend and R. Ahlrichs, Chem. Phys. 7, 3297, 2005.
[3] D. Rappoport and F. Furche, J. Chem. Phys. 133, 134105, 2010.
[4] M. Y. Hussaini and T. A. Zang, Annu. Rev. Fluid Mech. 19, 339–367, 1987.
[5] A. H. Bhrawy and M. M. Al-Shomrani, Adv. Differ. Equ. 2012, 8, 2012.
[6] I. Shavitt and R. J. Bartlett, Cambridge University Press, 2009.
[7] X. Li and C. Xu, SIAM J. Numer. Anal. 47, 2108–2131, 2009.
[8] R. Bauernschmitt, M. Haeser, O. Treutler, and R. Ahlrichs, Chem. Phys. Lett. 264, 573–578, 1997.
[9] D. Rappoport and F. Furche, J. Chem. Phys. 122, 064105, 2005.
[10] Y. Saad: Iterative methods for sparse linear systems, SIAM, 2003.
[11] V. Simoncini and D. B. Szyld, Numer. Linear Algebra Appl. 14, 1–59, 2007.
[12] F. Furche, B. T. Krull, B. D. Nguyen, and J. Kwon, J. Chem. Phys. 144, 174105, 2016.
[13] M. Metcalf, J. Reid, and M. Cohen: Modern Fortran Explained, Oxford University Press, 2013.

Acknowledgements

This poster is based upon work supported by the National Science Foundation under OAC-1835909 (CSSI Elements)
N. G. was funded by the UCOP UC-HBCU Pathways grant awarded to V. Ara Apkarian under CHE-1414466.

Public Repository at: gitlab.com/libkrylov F2.✓