The algebraic structure: properties of the coefficient matrix the adjoint system related to an optimal control approach in semiconductor design - and an efficient parallel realization in petsc

Maubach, J.M.L.

Published: 01/01/2005

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal
The algebraic structure/properties of the coefficient matrix the Adjoint system related to an optimal control approach in semiconductor design – and an efficient parallel realization in petsc

dr. J.M.L. Maubach
Eindhoven University of Technology
Faculty of Mathematics and Computer Science
CASA/Scientific Computing

December 9, 2005

Contents

1 Introduction 1

2 The adjoint problem 2
   2.1 The equations ................................................. 2
   2.2 The matrix macro-block-form ................................ 3
   2.3 Micro-block or macro-block Jacobian matrix ............. 4
   2.4 The finite difference approximations ...................... 5
   2.5 Norm for convergence ...................................... 7
   2.6 Minimum discrete grid dimensions ......................... 7
   2.7 The tensor and uniform grids ............................. 7

3 The Jacobian dF 8
   3.1 Derivation from operator F .................................. 8
   3.2 Diagonal dominance .......................................... 9
   3.3 Symmetry ..................................................... 11
   3.4 Positive definiteness ....................................... 11
   3.5 Assembly .................................................... 12
   3.6 Parallel multiplication .................................... 13
1 Introduction

This document examines the structure of the adjoint system in the paper [1]. This system consists of linear equations of which one is a vector-valued problem, with a special matrix...
block-structure.

The proposed discretizations are finite difference discretizations on tensor structured grids – rather than the standard finite difference stencils for uniform grids. This facilitates possible better solutions near the thin local contact zones in diodes and transistors.

This paper shows how to discretize the individual linear operator parts (which lead to the blocks of the matrix block-structure) such that the the vector-valued problem will have as much desirable properties (diagonal dominant, positive definite, symmetric, micro/macro-block, etc.). The proposed discretizations are standard (or can be derived in a standard manner) and have $O(h^p)$ truncation error. However, not all $O(h^p)$ discretizations lead to the desirable properties, but this paper derives methods which do.

The special properties are required for an effective solution of the linear scalar and vector-valued problems. The latter problem has a complex non-trivial block-structure and without a proper discretization method no solution will be possible for fine computational grids.

Petsc is used to facilitate parallel computation of the solution which is a must for production purposes and commercial activities. It is also used as the basis of NumLab (Scientific Computing Group, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands). NumLab facilitates the rapid application development.

The remainder of this document describes the construction of the discrete operators from suitable parts, comments at properties of the individual block-operators and properties of petsc which determine the implementation (possibilities).

This paper does not proof special properties of the block-structured matrix related to the vectorial sub-problem of the adjoint system. The reason has been the lack of time – the derivation of suitable discretizations and the implementation of a pilot program had to be done within a 1 month period.

Future research and use is anticipated to take place within the European project COMFON and to take place within the Eindhoven NumLab framework. And will contain the examination of the properties of the vector valued sub-problem of the adjoint problem.

2 The adjoint problem

2.1 The equations

This section formulates the adjoint problem as a linear problem to be solved. There are three linear subproblems (see [1]) of which we consider the largest problem in detail.
Written in more or less the convection-diffusion form, the equations (14) – (16) from A.M.Anile/C.Grado are (first index is the component, second one the equation, non-standard form) reformulated into:

\[
\begin{align*}
-a_{11} \Delta \xi_1 - \nabla a_{11} \nabla \xi_1 & + b_{11} \nabla \xi_1 \\
-a_{21} \Delta \xi_2 - \nabla a_{21} \nabla \xi_2 & + b_{21} \nabla \xi_2 + c_{21} \xi_2 + c_{31} \xi_3 = 0 \\
-a_{12} \Delta \xi_1 - \nabla a_{12} \nabla \xi_1 & + b_{12} \nabla \xi_1 \\
-a_{22} \Delta \xi_2 - \nabla a_{22} \nabla \xi_2 & + b_{22} \nabla \xi_2 + c_{22} \xi_2 + c_{32} \xi_3 = 0 \\
-a_{33} \Delta \xi_3 + b_{13} \nabla \xi_1 & + b_{23} \nabla \xi_2 + c_{23} \xi_2 + c_{33} \xi_3 = 0
\end{align*}
\]

where:

\[
\begin{align*}
a_{ji} & = D_{ji}(w, V) \\
a_{33} & = \lambda^2 \\
b_{ji} & = \sum_{k=1,2} \frac{\partial d_{jk}}{\partial w_i} \nabla w_k (i, j = 1, 2) \\
c_{2i} & = -\frac{\partial q(w, V)}{\partial w_i} \\
c_{3i} & = +\frac{\partial n(w, V)}{\partial w_i}
\end{align*}
\]

Here we use that the coefficient functions \(d_{ij}, q\) and \(n\) depend on \([w_1, w_2, w_3 = V]\).

The other 2 equations to be solved are

\[-\Delta \phi + \phi = 0\]

with mixed homogeneous dirichlet bcs and non-homogeneous neumann bcs, and

\[-\Delta C = -(\Delta \bar{C} + \gamma^{-1} \xi V)\]

with homogeneous dirichlet bcs. Discretized in the standard manner, the discretized operators are symmetric positive definite and the problems can be solved in \(O(1)\) time – with (algebraic) multigrid.

### 2.2 The matrix macro-block-form

The adjoint sub-problem [1] has blockstructure form:

\[
\begin{array}{cccc}
+&+&+&+ \\
| & | & | & |
\end{array}
\]

Input:

1. \(I_g\) – function on part of \(\partial \Omega\) – or function on \(\Omega\).
We choose for a function on the domain to be read in from a file so in future versions, one can iterate over $I_g$ as well.

Output: $\phi$ – function on $\Omega$

The adjoint sub-problem [2] has blockstructure form:

\[ +---+---+---+ +--------+ +----+
| \xi_{w1} | | |
\[ +---+---+---+ +--------+ +----+
| \xi_{w2} | = | |
\[ +---+---+---+ +--------+ +----+
| \xi_{V} | | \phi|
\[ +---+---+---+ +--------+ +----+

Input:
1. $w_1$ – function on $\Omega$.
2. $w_2$ – function on $\Omega$.
3. $w_3 = V$ – function on $\Omega$.
4. $\phi$ from sub-problem [1]

Output:
1. $\xi_{w1}$ – function on $\Omega$.
2. $\xi_{w2}$ – function on $\Omega$.
3. $\xi_V$ – function on $\Omega$.

The adjoint sub-problem [3] has blockstructure form:

\[ +--+ +--+ +--------+
\[ |C| = |\xi_{V}| |
\[ +--+ +--+ +--------+

In all cases – which is a (non-important) petsc limitation:

In order to use a “vector-valued” amount of dofs per grid point, all matrix blocks must be square. This can be achieved as follows. All blocks have identical size if the essential conditions are not used to eliminate variables. Hence: Dirichlet conditions lead to a dof (!) where in the Hessian the related diagonal entry is put to 1, and in the operator the related entry is put to zero. The initial guess for this guess should contain the correct Dirichlet value. Then all Newton updates will keep this value invariant.
2.3 Micro-block or macro-block Jacobian matrix

The difference is in the order of the degrees of freedom. Macro-block vector: First all
degrees of freedom related to 1 solution component, next all dofs related to the next
Micro-block vector: First the first degree of freedom of all components, next the second
dof of all components, etc.

The difference is the difference between the solution of $Ax = b$ and $PAP^T y = Pb$ where $P$
is a permutation matrix.

Iterative solvers whose speed of convergences is based on the spectrum will converge in the
same amount of iterations for the micro and macro case – but round-off could lead to a
(small) difference in iteration count.

However as soon as a ILU(0) preconditioner is used, the amount of iterations will depend
on micro or macro (because the different order leads to different (not-permutation-related)
ILU(0) factors.

Must use -mat_view in petsc to determine whether it is micro or macro block order:
Petsc turns out to use the micro-block order (as could be expected from the fill routine
MatSetValuesStencil()).

2.4 The finite difference approximations

Let $s$ be the solution of the pde $F(s) = 0$ (which satisfies the BCs). Let $u$ be the approxi-

mation for which $F_h(u) = 0$ (which satisfies the BCs). Where $F$ contains derivatives, $F_h$
contains the finite difference approximations. Then:

1. on a uniform grid, the central difference stencil for 2nd derivative: $u \mapsto [-u_{i-1} +
2u_i - u_{i+1}]/h^2$ introduces an error $O(h^2)u^{(4)} \Rightarrow$ for all up to cubic degree polynomials
$p$, $F(p) = F_h(p)$ (exact) $\Rightarrow ||s - u||_{max} = O(h^2)$ and $||s - u||_2 = O(h)$ for $h \to 0$

2. on a tensor grid, the central difference stencil for 2nd derivative is as follows: assume
intervals $e_i$ with limit points $u_{i-1}$ and $u_i$ has width $h_i = x_i - x_{i-1}$ (the program
code uses $h_{left}$ and $h_{right}$ instead of $h_i$ and $h_{i+1}$). Then $u \mapsto [-h_{i+1}u_{i-1} + (h_i +
h_{i+1})u_i - h_i u_{i+1}]/[h_i \cdot h_{i+1} \cdot (h_i + h_{i+1})/2]$ the error is
$[(h_i^3 h_{i+1}^3 - h_i^3 h_{i+1})/6]/[h_i \cdot h_{i+1} \cdot (h_i + h_{i+1})/2] \cdot u^{(3)}$ the error is $O(h)u^{(3)}$ if $h_i^3 h_{i+1} - h_i^3 h_{i+1} \neq 0 \Rightarrow$ for all up to
quadratic degree polynomials $p$, $F(p) = F_h(p)$ (exact) $\Rightarrow ||s - u||_{max} = O(h)$ and
$||s - u||_2 = O(1)$ for $h \to 0$. Note: With the 3 points (and requirements that the
contribution of the stencil is $0 \cdot u_i$, $0 \cdot u_i^{(1)}$ and $1 \cdot u_i^{(2)}$) no more accurate stencil is
possible.

3. on a uniform grid, the forward difference stencil for 1-st derivative: $u \mapsto [u_{i+1} - u_i]/h$
the error is $O(h^1)u^{(2)} \Rightarrow$ for all up to linear degree polynomials $p$, $F(p) = F_h(p)$
(exact) $\Rightarrow ||s - u||_{max} = O(h)$ and $||s - u||_2 = O(1)$ for $h \to 0$
4. on uniform grid, the central difference stencil for 1-st derivative: $u \mapsto \left[u_{i+1} - u_{i-1}/(2h)\right]$ the error is $O(h^2)u^{(3)} \Rightarrow$ for all up to quadratic degree polynomials $p$, $F(p) = F_h(p)$ (exact) $\Rightarrow \|s - u\|_{\text{max}} = O(h^2)$ and $\|s - u\|_2 = O(h)$ for $h \to 0$

5. on uniform grid, upwind differences are such that the diagonal dominance is kept invariant:

\[
\begin{align*}
  u'(x) &= \lim_{h \to 0} \frac{[u(x + h) - u(x)]/h}{[u(x + h) - u(x)]/h} \\
  &\approx \frac{[u(x + h) - u(x)]/h}{[u(x + h) - u(x)]/h} \\
  &= \frac{[u_{i+1} - u_i]/h}{[u_{i+1} - u_i]/h} \\
  &= \frac{[u(x) - u(x - h)]/h}{[u(x) - u(x - h)]/h} \\
  &= \frac{[u_{i} - u_{i-1}]/h}{[u_{i} - u_{i-1}]/h}
\end{align*}
\]

The first approximation above would add $-1/h$ to the diagonal and should be used for the approximation of $au'(x)$ where $a < 0$. The second approximation above would add $+1/h$ to the diagonal and should be used for the approximation of $au'(x)$ where $a > 0$. the error is $\|a\|_{\text{inf}} \cdot O(h)u^{(2)} \Rightarrow$ for all up to linear degree polynomials $p$, $F(p) = F_h(p)$ (exact) $\Rightarrow \|s - u\|_{\text{max}} = O(h)$ and $\|s - u\|_2 = O(1)$ for $h \to 0$

6. on tensor grid, the forward difference stencil for 1st derivative: $u \mapsto \left[u_{i+1} - u_i/(h_i + h_{i+1})\right]$ the error is $O(h^2_i)u^{(2)} \Rightarrow$ for all up to linear degree polynomials $p$, $F(p) = F_h(p)$ (exact) $\Rightarrow \|s - u\|_{\text{max}} = O(h)$ and $\|s - u\|_2 = O(1)$ for $h \to 0$

7. on tensor grid, the central difference stencil for 1-st derivative: $u \mapsto \left[u_{i+1} - u_{i-1}/(h_i + h_{i+1})\right]$ the error is $O(h_i^2)u^{(2)}$ if $h_{i+1} - h_i \neq 0 \Rightarrow$ for all up to linear degree polynomials $p$, $F(p) = F_h(p)$ (exact) $\Rightarrow \|s - u\|_{\text{max}} = O(h)$ and $\|s - u\|_2 = O(1)$ for $h \to 0$

8. on tensor grid, a 3 point central $\left([u_{i-1}, u_i, u_{i+1}]\right)$ difference stencil for 1-st derivative: let $h_i = x_i - x_{i-1}$ and $h_{\text{cubed}} = \frac{1}{2}h_i h_{i+1}(h_{i} + h_{i+1})$. Then $u \mapsto \left[(-\frac{1}{2}h_i^2/h_{\text{cubed}})u_{i-1} + (\frac{1}{2}h_i^2 h_{i+1})/h_{\text{cubed}})u_i + (\frac{1}{2}h_{i+1}^2/h_{\text{cubed}})u_{i+1}\right]$. Because the coefficients are $O(h^{-1})$ the error is of the order $O(h^2)u^{(3)}$. Note that for $h_{i+1} > h_i$ we risk to lose diagonal dominance.

9. on tensor grid, a 3 point right-oriented $\left([u_i, u_{i+1}, u_{i+2}]\right)$ difference stencil for 1-st derivative: let $h_{i+1} = x_{i+1} - x_{i}$ and $h_{i+2} = x_{i+2} - x_{i+1}$ and $h_{\text{cubed}} = h_{i+1} h_{i+2}(h_{i+1} + h_{i+2})/2$. Then $u \mapsto \left[(-\frac{1}{2}(h_{i+1} + h_{i+2})^2 + (\frac{1}{2}(h_{i+1})^2)/h_{\text{cubed}})u_i + ((\frac{1}{2}(h_{i+1} + h_{i+2})^2)/h_{\text{cubed}})u_{i+1} + (-\frac{1}{2}(h_{i+1})^2)/h_{\text{cubed}})u_{i+2}\right]$. Because the coefficients are $O(h^{-1})$ the error is of the order $O(h^2)u^{(3)}$ whence exact for up to 2-nd degree solutions. For an equidistant grid where $h_{i+2} = h_{i+1} = h$ we find

\[
  u \mapsto \left[-3u_{i} + 4u_{i+1} - u_{i+2}\right]/h
\]

Because the term $-3/h < 0$ as all real $h$ we find that straight forward application of this stencil in a derivative approximation will decrease the diagonal value. Hence, for
the approximation of Neumann BCs, we must use derivative approximation of the form: \(-([\text{stencil}] - \text{exact derivative}) = 0\) and \textbf{not} \((\text{stencil} - \text{exact derivative}) = 0\)

10. on uniform/tensor grid, the central difference stencil for 0st derivative, the error is zero \(\Rightarrow\) for all degree polynomials \(p\), \(F(p) = F_h(p)\) (exact)

11. These conclusions hold also for constant-coefficient-weighted-stencils \(\Rightarrow\) so for constant coefficient problems.

12. Assume we want to solve \(F(u) = 0\). Assume that if the Jacobian \(dF\) is positive definite. Assume one wants to solve the related transient problem: either \(\partial u/\partial t + F(u) = 0\) or \(\partial u/\partial t - F(u) = 0\). To keep properties such as positive definite, use \(\partial F/\partial t + F(u) = 0\). Example: \(F(u) = -\Delta u + u\). Then for the transient problem, solve \(\partial u/\partial t - \Delta u + u = 0\).

13. Note that the above problem also occurs when a root \(x\) of \(F\) must be translated into a fixed point \(x\) of \(G\). The amount of possible \(G\)’s is not finite (such as \(G(x) = x + F(x)\), \(G(x) = x - F(x)\), \(G(x) = H(x - F(x))\) for non-singular \(H\), etc.) but in order to have properties of \(F\) inherited by \(G\) one must choose a proper function \(G\).

For all stencils above, the stencil width is one – see petsc manual for def. of stencil width.

Other stencils exist: See for instance the central difference stencil for 2nd derivative of the form \(\nabla \cdot a \nabla u\) with potential non-smooth coefficient \(a\) (see Y. Saad).

### 2.5 Norm for convergence

Proof of existence of solution for finite difference operators \(F_h\) is related to the fact that their jacobian \(\partial F_h\) is monotone – which causes the truncation error to be equal to the approximation error. This standard proof uses the max-norm – and not the 2-norm. (recall 2-norm and max-norm differ a factor \(\sqrt{n}\) in \(\mathbb{R}^n\) whence for 2-dimensional FDM, 2-norm and max-norm differ a factor \(h\).

### 2.6 Minimum discrete grid dimensions

The minimum grid sizes in all coordinate directions are determined from the stencils. For instance: A 3-point stencil will require a grid size of at least 3. The stencils which influence the grid size are:

1. differential equation stencil(s)
2. neumann condition stencil(s)
3. stencils used to obtain the coefficients in the differential equation in the case that exact differentiation is not possible

Note: These stencils also determine the ”width” parameter for the petsc distributed array.
2.7 The tensor and uniform grids

The tensor grids are constructed as follows. Assume $\Omega = (0,1)^k$. Then map uniform to non-uniform (tensor) coordinates with a map $f: [0,1] \mapsto [0,1]$ – identical in all coordinate directions.

Example: Make $f$ such that a uniform grid is obtained. Take $f = \text{tensor_uniform}$: We use $f(x) = x$ to obtain a uniform grid.

Example: Make $f$ such that a finer grid is obtained at points 0 and 1. $f = \text{tensor_bubble}$: We require

- $f^{(k)}(0) = f^{(k)}(1) = 0$ for $k = 1$ and
- $f(0) = 0$;
- $f(1) = 1$.

(and anticipate that $f(1/2) = f(1/2)$). This leads to

$$f(x) = 3 \cdot x^2 - 2 \cdot x^3$$

. With this function, $[0,1/4,1/2,3/4,1]$ is mapped to (rounded) $[0, 0.15625, 0.5, 0.84375]$.

The use of petsc distributed arrays (da) support for a weak form of a tensorgrid with grid points $\{(x,y,z)_{i,j,k}\}_{i,j,k}$. This is more general than the tensorgrid form $\{(x_i,y_j,z_k)\}_{i,j,k}$. Possible support is for stencils (FDM, but also FVM and FEM if rewritten into stencil format).

Note that the domain is a $[a,b] \times [c,d] \times [e,f]$. Other domains can be handled – up them into a box which surrounds the domain and limit the stencil application to the domain + extras. However, all dofs in the box take part in all vector and matrix operations (so is expensive if the domain contains much less dofs than is surrounding box does. Up to fast multigrid solvers can be used.

3 The Jacobian $dF$

3.1 Derivation from operator $F$

Assumptions:

- $F$ is a vector valued function with components $f_i$
- $U$ is a vector valued with components $u_p$
- $S$ is a vector valued with components $u_p$
• the pde is in two dimensions and vector component $k_p$ has entries $v_{p,j,i}$

• $f_1$ and $f_2$ are defined as follows:

$$
f_1 &= (-\Delta_h u_1 + \partial_x^h u_2) - (-\Delta s_1 + \partial_x s_2), \\
f_2 &= (-\Delta_h u_2 + \partial_x^h u_1) - (-\Delta s_2 + \partial_x s_1),
$$

where in more detail

$$
f_{1,j,i} = \\
\begin{pmatrix}
-u_{1,j-1,i} \\
-u_{1,j+1,i} \\
-u_{1,j,i-1} \\
-u_{1,j,i+1} \\
+4 \cdot u_{1,j,i}/h^2 \\
+ \\
(-u_{2,j,i-1} \\
+u_{2,j,i}/h \\
) \\
- \\
( \\
-\partial_x^2 s_1(x_{i,j}) \\
-\partial_y^2 s_1(x_{i,j}) \\
+\partial_x s_2(x_{i,j}) \\
)
\end{pmatrix}
$$

Now the Jacobian $dF$ in row $(p,j,i)$ and column $(q,l,k)$ will contain $\partial_{q,l,k} f_{p,j,i}$. In most cases this will be zero. For $f_1 (p = 1)$ we find possible non-zero values for row $(1,j,i)$ in some of these columns $(q,l,k)$:

$$
q = 1 \\
l \in \{j - 1, j, j + 1\} \\
k \in \{i - 1, i, i + 1\}
$$

and

$$
q = 2 \\
l = j \\
k \in \{i, i + 1\}
$$

For instance: row $(1,j,i)$ and column $(2,j,i-1)$ contains the number (see $f_1$): $-1/h$.

### 3.2 Diagonal dominance

The diagonal element of row $(p,j,i)$ is at the column $(p,j,i)$ for all $p,j,i$. All other columns $(q,l,k) \neq (p,j,i)$ are off-diagonals for row $(p,j,i)$. In the example above where $p = 1$ we
find the columns

\[(1, j, i) : 4/h^2 \quad \text{(diagonal)}\]
\[(1, j, i - 1) : -1/h^2 \quad \text{(off-diagonal)}\]
\[(1, j, i + 1) : -1/h^2 \quad \text{(off-diagonal)}\]
\[(1, j - 1, i) : -1/h^2 \quad \text{(off-diagonal)}\]
\[(1, j + 1, i) : -1/h^2 \quad \text{(off-diagonal)}\]
\[(2, j, i) : 1/h \quad \text{(off-diagonal)}\]
\[(2, j, i - 1) : -1/h \quad \text{(off-diagonal)}\]

Here we see – which holds in general – that the extra off-diagonal components due to the second solution component lead to the loss of diagonal dominance. This is true in general.

However – also in general – we obtain a macro-block matrix (assume 2 components)

\[dF = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\]

where each block could be diagonal dominant etc.. We could make use of the properties of the individual blocks with the use of a block-structure iterative-solvers – but these do not exists for the general case.

Note: There are famous algebraic transformations which can make a block matrix have a positive definite symmetric part. For instance: The Stokes equations:

\[\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}\]

relate to a non-positive definite matrix but the equivalent equations

\[\begin{pmatrix} A & B \\ -B^T & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}\]

have a symmetric part

\[\begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}\]

which is positive definite if \(A\) is.

Example: Let \(A\) be a positive definite square matrix and let \(B\) be a rectangular matrix. Then

\[\begin{pmatrix} A & B \\ -B^T & S \end{pmatrix}\]

has a positive definite symmetric part. The matrix \(B = I\) (identity) is option 2 of ex8.c for the off-diagonal part.
3.3 Symmetry

In the non-discretized case the Jacobian is symmetric if and only if $\frac{\partial f_q}{\partial p} = \frac{\partial f_p}{\partial q}$ for all components $p$ and $q = 1, \ldots, \text{nb\_components}$. 

In the discretized case the same relation holds, but for the discretized functions $f_j$ and $f_i$: $\frac{\partial f_j}{\partial i} = [dF]_{i,j}$ must be $\frac{\partial f_j}{\partial i} = [dF]_{j,i}$ where $i$ and $j$ are the indices (such as $(p, j, i)$ and $(q, l, k)$). In the example, we find

$$\frac{\partial f_{2,j,i-1}}{\partial f_{1,j,i}} = -1/h$$

and

$$\frac{\partial f_{1,j,i} f_{2,j,i-1}}{\partial f_{1,j,i}} = \frac{\partial}{\partial f_{1,j,i}} \left[ \begin{array}{c}
\frac{-u_{2,j-1,i-1}}{h^2} \\
-u_{2,j+1,i-1} \\
-u_{2,j,i-2} \\
-u_{2,j,i} \\
+4 \cdot u_{2,j,i-1})/h^2 \\
+ (-u_{1,j,i-2} \\
+u_{1,j,i-1})/h \\
\end{array} \right]$$

Here we substituted index $(2, j, i-1)$ for the index $(2, j, i)$ in the definition of $f_{2,j,i}$. Hence, the Jacobian related to $F = [f_1, f_2]$ is not symmetric.

In general, if the non-discretized Jacobian is not symmetric then the discretized is neither. If the non-discretized Jacobian is symmetric, then the discretized might be – when the difference stencils are chosen with care.

3.4 Positive definiteness

Krylov subspace linear solvers require (the symmetric part of) the coefficient matrix $A$ of $Ax = b$ to be positive definite.
For a matrix obtained with finite differences or volumes (not finite elements) this is best shown via: (1) diagonal dominant + symmetry ⇒ symmetric positive definite.

Or show that the symmetric part is positive definite via: (2) symmetric part is diagonal dominant + symmetric (automatic) ⇒ symmetric part is positive definite.

### 3.5 Assembly

Use `FormFunctionJacobian(...)`, where `PdF` contains the matrix to be used as a preconditioner.

In petsc one can both fill `dF` exor `PdF` and default (when not filled out) the other matrix will be identical to the filled out matrix.

However, pets suggests (manual page 87, section 5.1.2) that one fills out the (analytically computed discrete) Jacobian in `PdF`. Then the option `-snes_mf_operator` can be used to approximate `dF` with finite differences and to use the analytical Jacobian in `PdF` as the basis of a preconditioner.

There are related `ksp` reasons to fill out the preconditioner’s base (matrix) and not the matrix itself. Petsc manual page 167 mentions that some preconditioners whould profit (efficiency) if the preconditioner can alter the matrix vector multiplication, so the preconditioner should be filled out. Personal (maubach) comment: I don not understand this. To fill out just one and not the other means that petsc must do something to create at least the other and it does not tell what. I think the whole issue is implementational and should not surface what soever. Petsc better explained what happens if one (matrix) is filled out and the preconditioner’s base matrix not – or vice versa.

The Jacobian is filled out with the use of stencils:

- `MatSetStencil()`
- `MatSetValuesStencil()`
- `MatStencil()`

You MUST create the matrix NOT with `MatCreate` but YES with `DAGetMatrix` (see `snes/ex14.c`).

Note: In petsc examples which use `MG` this fact (the required use of `DAGetMatrix`) is hidden. Reason: The `dmmg` context does not get a matrix from the user, it instantiates its own to call the `FormJacobian` (and it does so with its da, which is passed to `dmmg`).
An example which fills the da-matrix with `MatSetValuesStencil()` for the scalar case: 
`src/snes/examples/tutorials/ex18.c.html`.

There exists no example which fills a da-matrix with `MatSetValuesStencil()` for the scalar case. In order to find out that the "c"-field in the MatStencil is indeed the component number, see the implementation file `src/mat/interface/matrix.c` line 719.

### 3.6 Parallel multiplication

Example: Solve

\[
Au = b
\]

where

\[
A = \begin{pmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{pmatrix}
\]

and

\[
b = \begin{pmatrix} 0 \\ 0 \\ 4 \end{pmatrix}
\]

whence

\[
u = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}
\]

We will solved the system of equations with an iterative solution method which (standard) uses:

1. vector space operations \(v + w, \alpha \cdot v\);
2. matrix vector multiplications \(Av\).

Assume we use of two processors, and distributed array vector and matrices. Then petsc vector layout turns out to be

1. processor 0:

\[
A = \begin{pmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{pmatrix}
\]

and

\[
b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}
\]

whence

\[
u = \begin{pmatrix} 1 \\ 2 \end{pmatrix}
\]
2. processor 1:

\[ A = \begin{pmatrix} 0 & -1 & 2 \end{pmatrix} \]

and

\[ b = (4) \]

whence

\[ u = (3) \]

Assume that \( A \) is distributed (petsc-wise) across 2 processors. Note that

\[ A(Au) \]

will need communication across processors – even if each processor would store the entire vector \( u \). This is due to the fact that \( v = (Au) \) must also be available to both processors for the multiplication \( A(Au) = Av \). Because \( Au \) is distributed across 2 processors communication is needed to get all values available at both processors.

4  Eindhoven/Catanian/Petsc

4.1 The examples

Explanation of the abbreviations used in the table of examples:

- Amount of processors: \( P: 1, 2, \ldots, > 0, > 1, \ldots \)
- Spatial dimension: \( D: 1, 2, \ldots, > 0, > 1, \ldots \)
- Mesh: \( M: (U)niform; (T)ensor \)
- Documentation: \( T: \) poor/ok/best are -/o/+ 
- Equation(s): Equation(s)
- Boundary conditions: \( BCS: D/N/R/? \) are Dirichlet/Neumann/Robin/undetermined 
- Discretization: \( F: \) finite Differences/Elements/Volumes are D/E/V, S is staggered 
- Matrix Free: \( MF: \) Matrix/No-Matrix are M/F 
- Multigrid: \( MG: \) MultiGrid/No-Multigrid are M/N 

The table of examples:
<table>
<thead>
<tr>
<th>P</th>
<th>D</th>
<th>M</th>
<th>T</th>
<th>Equations</th>
<th>BCS</th>
<th>F</th>
<th>MF</th>
<th>MG</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>U</td>
<td>-</td>
<td>$u'' + u^2 = f$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>snes/ex2.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>1</td>
<td>U</td>
<td>-</td>
<td>$u'' + u^2 = f$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>snes/ex3.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>1</td>
<td>U</td>
<td>-</td>
<td>$u'' + u^2 = f$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>snes/ex3.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>1</td>
<td>U</td>
<td>-</td>
<td>$\partial_t u = \frac{u''}{u'' + u^2}$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>ts/ex2.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>U</td>
<td>-</td>
<td>$-\Delta u - \lambda \exp(u) = 0$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>snes/ex5.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>1</td>
<td>U</td>
<td>-</td>
<td>$u'' + u^2 = f$</td>
<td>D</td>
<td>D</td>
<td>F</td>
<td>N</td>
<td>snes/ex6.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>3</td>
<td>U</td>
<td>-</td>
<td>$-\Delta u - \lambda \exp(u) = 0$</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>snes/ex14.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>U</td>
<td>-</td>
<td>$-\nabla \cdot (u^{5/2} \nabla u) = 0$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>snes/ex18.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>U</td>
<td>-</td>
<td>$-\Delta u - \partial_y q$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>snes/ex19.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>3</td>
<td>U</td>
<td>-</td>
<td>$-\nabla \cdot (u^{5/2} \nabla u) = 0$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>snes/ex20.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>U</td>
<td>-</td>
<td>$-\nabla \cdot (1 + \sqrt{\nabla u})^{1/2} \nabla u = 0$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>snes/ex25.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>U</td>
<td>-</td>
<td>$-\Delta u - \partial_y q$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>snes/ex30.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>T</td>
<td>+</td>
<td>$-a \Delta u + b \nabla u + cu = f$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>maub/ex6.c</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>2</td>
<td>T</td>
<td>+</td>
<td>$-a \Delta u + b \nabla u + cu + v = f$</td>
<td>DN</td>
<td>D</td>
<td>M</td>
<td>N</td>
<td>maub/ex9.c</td>
</tr>
</tbody>
</table>

As a rule all > 1 dimensional uniform mesh examples use distributed arrays (DAs).

Specific names for the equations:

1. The Bratu (Solid Fuel Ignition) problem: $-\Delta u - \lambda \exp(u) = 0$
2. The time-dependent Bratu (Solid Fuel Ignition) problem: $\partial_t u - \Delta u - \lambda \exp(u) = 0$
3. Nonlinear Radiative Transport PDE: $-\nabla \cdot (u^{5/2} \nabla u) = 0$
4. The minimal surface problem: $-\nabla \cdot (1 + \sqrt{\nabla u})^{1/2} \nabla u = 0$
5. The driven cavity problem in a velocity-vorticity formulation:

\[
\begin{align*}
-\Delta u - \partial_y q &= 0 \\
-\Delta v - \partial_x q &= 0 \\
-\Delta p + \nabla \cdot [u, q, v, q] - \nabla \cdot \partial_x T &= 0 \\
-\Delta T + pr \nabla \cdot [u, T, v, T] &= 0
\end{align*}
\]

6. Optimization problem: Minimize $F(u)$ such that $G(u) = 0$, i.e., set

\[
L(u, \lambda) = F(u) + \lambda^T G(u)
\]

and solve for

\[
\nabla_{u, \lambda} L(u, \lambda) = 0
\]

(see petsc examples snes/ex21.c -- snes/ex24.c)
7. Steady-state 2D subduction flow, pressure and temperature:

\[-\nabla p + \nabla \cdot [\eta(t, p, v)\nabla v + \nabla v^T] = 0\]
\[\nabla \cdot [u, w] = 0\]
\[\partial T/\partial t + \nabla \cdot [vT] - 1/\rho e \cdot \nabla^2 (T) = 0\]

for different functions \(\eta\)

8. Systems of convection-diffusion-like equations:

\[-a_1 \Delta u + b_1 \nabla u + c_1 u + v = f\]
\[-a_2 \Delta v + b_2 \nabla v + c_2 v - u = f\]

Remarks:

- In the maub/... examples, different-order finite difference stencils for the operator and the boundary conditions can be chosen – likewise different meshes (U or T). All options are explained.
- In some time-dependent examples, pseudo-time-stepping is used

Further examples exist for

1. Grad-Shafranov solver for one dimensional CHI equilibrium

4.2 The structure

All examples ex2.c – ex*.c do the same: A (non-)linear problem \(F(U) = 0\) is solved with the use of a non-linear solution method, which uses a krylov type linear solver and a preconditioner (all can be set with command line options). In particular, \(U \mapsto F(U)\) is a partial differential operator discretized with finite differences at a uniform or tensor grid. All examples are structured as follows:

1. DA (distributed array) set up;
2. Potential uniform tensor grid transformation;
3. Allocate vectors \(U, F\) and matrix \(\partial F x\)
4. Define: FormApproximation: Provide the first iterand \(U^{(0)}\) (non-petsc routine) FormSolution: Provide the solution \(S\) – if it exists (non-petsc routine) FormFunction: Provide \(U \mapsto F(U)\) (the residual vector) (petsc snes routine) FormJacobian: Provide \(U \mapsto dF(U)\) (the Jacobian matrix) (petsc snes routine)
5. Feed the allocated vectors/matrix and defined functions to SNES with:
ierr = SNESSetFunction(snes, F, FormFunction, &ctx); CHKERRQ(ierr);
ierr = SNESSetJacobian(snes, dF, dF, FormJacobian, &ctx); CHKERRQ(ierr);

6. Call SNESolve() to obtain $U = U^{(k)}$, the final approximation

7. Compare $||U - S||_{\text{max}}$

Different operators and solutions can be chosen with options (run the example with the -help option to see a list of the options) and are realized with the use of switches in the 4 $\text{Form*()}$ functions above.

Petsc can calculate $\partial F(U)$ with the use of finite differences. If $F$ is linear then this is exact. If petsc distributed arrays are used the determination of the non-zero structure of the (in our case approximate == exact) Jacobian follows from the da’s stencil-width and dimension. However, we fill out the Jacobian by hand for efficiency.

Right now the $U^{(0)}$ is zero except at the Dirichlet BCs points where its values are the Dirichlet values. The latter must hold for all initial solutions one wants to use. The construction here, with an initial solution zero inside and possible non-zero at the boundary is not a good initial guess because the H1 norm $\to \infty$ for the grid dimensions $n, m \to \infty$.

4.3 The structure of the petsc adjoint implementation

There are two distributed arrays (das): 1 with one dof and one with 3 dofs, stored inside ctx_scalar and ctx_vector.

Because there are three problems to solve (two scalar and one vector) there are

FormApproximation*
FormSolution*
FormFunction*
FormJacobian*

for $* = 1, 2, 3$.

In order to reduce the vector count the two scalar equations ($\text{Form*1 and Form*3}$) share the vectors U_scalar, F_scalar, dF_scalar and S_scalar. The vector equation ($\text{Form*2}$) has its private vectors U_vector, F_vector, dF_vector and S_vector.

The snes solvers are also just two: snes_scalar and snes_vector.

Inside the Form*1--3 functions, these vectors (U_scalar – S_vector) are still called U, F, dF and S.
The options of the scalar and vector solver mixed and a bit adjusted:

- **solution_scalar**: set scalar solutions (option used to be called solution)
- **solution_vector**: set vector solution (option used to be called solution)
- **operator**: set scalar operator F
- **diagonal**: set vector operator F diagonal part (related to dF)
- **off_diagonal**: set vector operator F off_diagonal part (related to dF)

5 **Petsc**

5.1 **Features of petsc**

- runs in linux + windows (under cygwin)
- runs serial and parallel based on mpi/lapack/blas
- has quite a few routines to start with
- more or less public domain (some routines are free, others are gpl, other are ...), for most part is a kind of a BSD licence.
- written in c/fortran whence simple to interface
- is the basis of linear algebra numlab (TU/e)
- for tensor grid (da) applications, 1 \( \rightarrow \) 3 dim is simple

Built on top of petsc (more or less public domain):

- superlu (LU decomposition serial and parallel)
- etc.

Built on top of petsc (commercial use restricted):

- hypre (preconditioners for unstructured grids); [http://www.llnl.gov/CASC/linear_solvers/](http://www.llnl.gov/CASC/linear_solvers/)

For documentation, see [2], [3], [4] and [5].
5.2 Petsc: What is simple, difficult and perhaps dangerous

To be short:

1. Simple: The solution of time-dependent non-linear PDEs
2. Difficult: The communication (input/output) with non-petsc
3. Difficult: Solution components with different geometrical dof layout
4. Dangerous: Undocumented GhostCoordinates support (see below).

The communication with non-petsc programs: It turns out that petsc VecLoad() can sole load vectors saved in petsc binary format with the VecView() format. Furthermore, da-vectors can not be saved/loaded due to a bug which causes a crash in the multi-processor mode. Hence, in order to read a vector into petsc one must write it with subsequent calls to PetscBinaryOpen() multiple PetscBinaryWrite() and PetscBinaryClose().

The examples make when needed two assumptions. Petsc does not ensure these assumptions hold – but offers no alternatives.

1. ghost coordinates are kept between different calls to DAGetGhostedCoordinates() (see the related section) – this is needed because the call to DAGetGhostedCoordinates() is just collective the first call.

2. a 1-component (dof) da and a 3-component (dof) da which use the same processor distribution $m \times n$ have the same index ranges per processor (see the section on communication between different dof das)

Elaborate work-arounds exists (see sections).

5.3 Petsc limited support for da-coordinates, a risk

The petsc routine DAGetGhostedCoordinates() is just the first call collective (i.e., then the ghost coordinates are shared and correct).

This means that our grid transformation from uniform -tensor would not be correct a next DAGetGhostedCoordinates() if we would just alter the local owned coordinates because the ghosted coordinates would not be corresponded a next call to DAGetGhostedCoordinates().

The solution is that all processors ALSO alter the local ghosted (not-owned) coordinates. These are remembered from one call to DAGetGhostedCoordinates() to the next. Petsc does not comment on whether this (remembering) is a feature – so we run a risk for the future.

Note that one can store $x_i, y_i, z_i$ etc in a global array and use that one. However, the version with DAGetGhostedCoordinates() is more matlab like.
5.4 Installation PETSC

See the tutorial or see pets installation page: http://www-unix.mcs.anl.gov/petsc/petsc-as/documentation/installation.html

Summarized – do not first copy and paste but read first:

Download petsc.tar.gz into

```
#!/bin/bash
mkdir -p ${HOME}/software/petsc
mv petsc.tar.gz ${HOME}/software/petsc
cd ${HOME}/software/petsc
tar -xzvf petsc.tar.gz
cd petsc-2.3.0
export PETSC_DIR=${HOME}/software/petsc/petsc-2.3.0
```

1]# if mpi is installed then must specify location
$PETSC_DIR/config/configure.py --with-mpi-dir=/opt/mpich/ch-p4mpd
export PETSC_ARCH=linux-gnu
make # DO NOT FIRST JUMP INTO src!!!

2] # if there is no mpi installed -- ensure internet connection:
$PETSC_DIR/config/configure.py --download-mpich
export PETSC_ARCH=linux-gnu
cd src
make all
make test

# update .profile: append
export PETSC_ARCH=linux-gnu
export PETSC_DIR=${HOME}/software/petsc/petsc-2.3.0
export PATH=/opt/mpich/ch-p4mpd/bin/:$PATH

5.5 The use of da-vectors with ghosted values

The petsc manual states that in order to use a global vector with ghost points one must proceed as follows (p43-44):

1. Assume the global vector was created with DACreateGlobalVector U
2. Create a local vector with `DACreateLocalVector L`

3. Use `DAGlobalToLocalBegin(Da, U, INSERT_VALUES, L) DAGlobalToLocalEnd(Da, U, INSERT_VALUES, L)`

4. Use `DAVecGetArray(Da, L, _L)`

5. Use the c-array `_L` which also contains the ghost points.

6. Use `DAVecRestoreArray(Da, L, _L)`

7. Use `DARestoreLocalVector(Da, L)`

5.6 The use of multi-component da-vectors

Petsc has no example which demonstrates the use of `MatSetValuesStencil` in combination with a `>1`-dof da. See example `ex9.c` to this end.

Multi-component da-vectors examples are also the da’s coordinates: The field of coordinates returned with `DAGetCoordinates` is of type `DACoord3d *` (in the 3-d case). Access the field with: `x = coors[k][j][i].x; y = coors[k][j][i].y; z = coors[k][j][i].z` Note that there is no `DACoord1d`.

5.7 make 1-component-da and multiple-component-da communicate

Petsc (see the manual) can not instantiate a 1-dof vector from a 3-dof da (just 3 dof-vectors can be instantiated from a 3-dof da). This is a problem because if one instantiate a 1-dof vector from a 1-dof da next to a 3 dof-vector from a 3-dof da it is possible that the vectors are not partitioned likewise: It is possible that the index ranges at the same processors are different.

Needed for fast communication:

1. different dof da’s use same amount of processors;
2. different dof da’s use the same physical processors;
3. different dof da’s use the same index range on the same physical processors.

The current program `ex10.c` uses a 1-component DA and a 3-component DA and ensures that (1) and (2) holds and aborts if (3) would not hold: In the case of different amounts or different physical processors, there would be processors which have a different index range for the variable.
Without the use of the two das, the user must himself translate the single-index saved non-da-vector into a three-index da-vector (for 3 dofs). This requires information about how the petsc da translates between multiple and single – which is not public.

The vector coefficients of a single da-vector are ordered (3 x 3 points) printed with PETSC_VIEWER_STDOUT_WORLD are as follows (see ex10.c and the petsc manual, page 46, Figure 9):

```
6 7 8
3 4 5
0 1 2
```

where coefficient 1 is related to (in the example) point (0.0, 0.0) and coefficient 9 is related to (1.0, 1.0). Here the coefficient $i$ is aligned with the $x$-axis and the coefficient $j$ is aligned with the $y$-axis.

Note that a da uses writes on global coefficients $[j][i]$ – but ONLY the local subset $[j][i]$ can be used – hence, a da does not provide the equivalent of the VecSetValues(....). If one wants to write to global $[j][i]$, one must go via a vector and do all index transformations by hand.

A VecScatter() seems not possible as it seems just possible between vectors allocated with the same routine (the DACreate1d() for instance ...).

### 5.8 The save/load of a da vector

In order to save/load vector in DA format, use:

1. PetscViewerBinaryOpen(...). Be sure to: Use option PETSC_FILE_CREATE if the file could not yet exist (should be default) Use option PETSC_FILE_WRONLY if it is sure that the file exists (do not use this) Do not use PetscViewerASCIIOpen() because DAView() can not work with it

2. DAView() to save the vector’s da VecView() to append the vector’s entries

3. DALoad() to load the saved vector’s da VecLoadIntoVector() to load the saved vector’s entries

Petsc feature: DAView() fails to work in combination with PetscViewerASCIIOpen(). One must use it in combination with PetscViewerBinaryOpen().

Petsc feature: petsc can not read vectors in ascii (!).

Petsc bug: DAView()/DALoad() fail to work for nb_processes > 1 – so can not be used in parallel mode.
5.9 The generation of excellent error messages

1. Just before function with name `a_name` insert (from the first column onwards):

   ```
   #undef __FUNCT__
   #define __FUNCT__ "a_name"
   ```

2. Inside this function, enclose the entire code (except declaration header) within

   ```
   PetscFunctionBegin;
   PetscFunctionReturn(0);
   ```

   Exception is `main()` where `PetscFunctionBegin()` must be omitted (done in `PetscInitialize()`)

3. Each petsc routine (except `PetscInitialize()`) should have form:

   ```
   ierr = PetscRoutine(...); CHKERRQ(ierr);
   ```

5.10 How to run the examples without mpi

Without mpi:

   `./ex6 -n 65 -m 65 -solution 1 -grid 2`

With mpi:

   `mpirun -n 3 ./ex6 -n 65 -m 65 -solution 1 -grid 2`

(or - depends on the type of mpi support):

   `mpirun -np 3 ./ex6 -n 65 -m 65 -solution 1 -grid 2`

5.11 How to check whether the ghosted information (grid points and solution values) work well

Simple: In the one processor case, ghost points do not exist - petsc introduces such points just for the multiple processor case. If `mpirun` gives the same result with `-n 1` and `-n 8` ghostpoint related calculations are ok. In case of failure, check whether the tensor grid also contains tensor-ghost points.
5.12 How to print data in Petsc

Just processor 0 prints: PetscPrintf(PETSC_COMM_WORLD, ...*...). All processors print: PetscPrintf(PETSC_COMM_SELF, ...*...), where ...*... stands for the standard printf format string + arguments.

5.13 How to get statistics about Petsc’s performance

Use the option(s):

- -log_info.
- -log_summary.
- -snes_view.
- -ksp_view.

5.14 How to find a Petsc example which contains a certain string/Petsc command

Question: Find a Petsc example which contains the string DALocalToGlobalBegin. Answer: Goto the Petsc source folder and execute:

```bash
for i in $(find . -name *.c); do echo FILE: $i; grep DALocalToGlobalBegin $i; done | less
```

5.15 How to debug a Petsc program when a segmentation violation (crash) occurs

If a program crashes or produces incorrect output. First, test if the failure occurs if run on one processor as well: mpirun -n 1 ./ex6. If so, test with 1 processor (serial mode):

1. mpirun -n 1 ./ex6 -start_in_debugger

2. In multiple processor mode: In the debug window, after the ”(gdb)” prompt, enter: cont;

3. Alternative in single processor mode 1: In the debug window, after the ”(gdb)” prompt, enter: run;

4. Alternative in single processor mode 2: When prompted to answer ”Start it from the beginning? (y or n)” enter: y

5. To get the chain of called routines, enter: backtrace

6. To print lines of code around line number N which caused error, enter: list N

25
7. To print current variable values, for instance of variable \( i \), enter: `print i`

Because petsc buffers stdout output for better results, it is possible to receive an `MPI_Abort()` in the middle of output (!) and that some output is lost when the program comes to a halt. So, some output could be missing.

5.16 How to proceed if the iterative solver does not converge as desired

If snes does not converge as desired, attempt:

1. `-snes_ls basic/basic_nonorms/quadratic/cubic`: (a different non-linear solver)
2. `-snes_monitor`: monitor residuals
3. `-snes_xmonitor`: monitor residuals graphically

Compare the Jacobian in `FormJacobian` with the finite difference Jacobian:

1. `-snes_type test`: compare analytical and fd Jacobian
2. `-snes_type test_display`: show difference Jacobian and fd Jacobian
3. `-snes-mf_operator`: use the finite difference Jacobian as Jacobian and the one from `FormJacobian` as the base for a preconditioner – note: special care of the matrices to be filled out in the `FormJacobian` is required. See the petsc manual – which is not too clear (precise) on this topic.

*If the problem is linear and the non-linear solver does not converge in 1 iteration – but in 2 or 3 iterations, then ensure that the linear systems are solved more accurate, for instance with option: `-ksp_rtol 1.e-12`.*

5.17 Strengths and limitations of petsc

1. limitations: documentation is poor to reasonable – as must be expected for a free product
2. Note that petsc has an idea of local stored data ("diagonal part" at `d nz`, p 55, and "off diagonal part" at page 55), related to the call of `MatCreateMPIAIJ()`
3. Petsc can not allocate with `DACreateGlobalVec` both a 1 component vector and a multi-component vector with the same processor layout. Either all vectors are multi-component, or all are 1-component (because the DA contains the dofs). For multi-component cases which need also single component functions (right hand side), it means that the multi-component solution must be allocated component per component with `VectorPack` and not as a single multi-component vector.
4. petsc da support turns out to be limited to maximal 3 dimensions (DAGetCorners() etc. support up to 3 dimensions). 4-d work is not possible

5. DAGetGhostedCoordinates() just is collective the first call. This means that one can not alter the predefined global grid and share the local redefinitions global. But it turns out that one can also alter the local ghosted coordinates which keep as are in between following calls to DAGetGhostedCoordinates().

6. petsc da grids can not be shared amongst das, it seems. expensive scalar and vectorial equations must be solved on the same domain after each other

7. petsc MatStencil support is limited to maximal 3 dimensions (due to its datastructure).

8. in FormFunction() and other Form*() one must make the ghost-values consistent with the use of an extra local vector. petsc does not offer a more efficient manner – to be more precise, in petsc one can not make a global vector consistent with respect to the ghost points. Does global DA vector does not store the ghostpoints on each local processor?

9. petsc da stencil uses that the stencil "width" is identical in all coordinate directions which might not be the case. could be more efficient

10. petsc stencil support is not optimal for Jacobians which come from pdes. There one has a fixed stencil which could be communicated to petsc in the DACreate*d() as follows (for a 5 points 2 dimensional case):

    | column 0: j-offset 0 i-offset 0 |
    | column 1: j-offset 0 i-offset -1 |
    | column 2: j-offset 0 i-offset +1 |
    | column 3: j-offset -1 i-offset 0 |
    | column 4: j-offset +1 i-offset 0 |

11. petsc uses c-casts to void * as in void(*)(void)

12. petsc does well: MatrixAssemblyEnd() signifies the end of the "write" access and start of the "read" acces for which the internal format can then be fine-tuned (first the internal format was finetuned for "write").

13. Use of a block part of a vector: Can be done if the block-vector U is created with the use of VecPack/VecPackCreateGlobalVector commands.

14. Let u and v be global Petsc vectors of type T. Then it is not possible to create a matrix related to a T = DAVector (what does MatDAADSetDA do?) or a T = VectorPack.
15. In snes-invoked-methods such as `FormFunction()`, in order to ensure that all ghost points are consistent, one must do a `DAGlobalToLocalBegin+DAGlobalToLocalEnd` (see `snes/ex3.c+ex14.c` remark in the man pages related to DA + `FormFunction`) because snes does not ensure this. It seems that petsc does not have a vector-method called "make-consistent". Would be nice if we did not have to code one. Note: It does not make (efficiency) sense to keep also the ghost values consistent in between $A \times x$ etc operations – it just make sense for operations which use the ghost points.

16. in `FormFunction()` and other `Form*()` one must make the ghost-values consistent with the use of an extra vector. petsc does not offer a more efficient manner – but could have.


18. not nice that `FormFunction()` depends on da commands if one uses da vector/matrix. It would be nicer if it did not depend on the kind of matrix/vector.

19. Block support for matrices is not ensured to work for all cases – no further comment is provided.

20. illogical:

   ```
   ierr = PetscPrintf(PETSC_COMM_WORLD, ...): just processor 0 prints:
   ierr = PetscPrintf(PETSC_COMM_SELF, ...): each local processor prints;
   ```

   but

   ```
   VecView(b, PETSC_VIEWER_STDOUT_WORLD): each processor prints local vector part;
   VecView(b, PETSC_VIEWER_STDOUT_SELF): is possible but leads to an error.
   ```


22. Petsc feature: can not `Load()` a vector prior written with View in ASCII format.
   Worse: Can not `Load()` an ASCII format vector at all (with a petsc command). The standard binary format is provided in the manpage of `VecLoad()`

23. Illogical: `DACoord2d` has no 1-dimensional analogon `DACoord1d` whence it is not clear whether it should be an `PetscScalar` or a `PetscReal` array.

24. `PetscViewerSetFormat()` does not permit the user the matlab-variant of "format long". It seems impossible to see all decimal digits in the stdout output?
25. petsc should explain what happens if in \texttt{FormJacobian} just the (matrix) is filled out and the preconditioner's base matrix not – or vice versa.

26. Vectors have no default length 0, the example below refuses to run:

\begin{verbatim}
VecCreate(PETSC_COMM_WORLD, &vvv);
ierr = VecView(vvv, PETSC_VIEWER_STDOUT_WORLD); CHKERRQ(ierr);
\end{verbatim}

27. Option \texttt{-ksp_view} mentions that the initial guess is the zero vector. This is not optimal for non-linear cases where the solution should be (in the limit) close to the previous solution in the outer non-linear loop!

28. If one sets \texttt{-ksp_rtol} to value $10^{-12}$ this tolerance inside the pc context is still value $10^{-5}$. Is this a bug? or do I have to change this tolerance in the pc context? How do I do this?

29. If the sequence (where no vector entries are set)

\begin{verbatim}
DACreateGlobalVector()
DAGlobalToLocalBegin()
DAGlobalToLocalEnd()
DAVecGetArray(da, F, &F); CHKERRQ(ierr);
DAVecRestoreArray(da, input_scalar_local, &input_scalar); CHKERRQ(ierr);
DARestoreLocalVector(da, &input_scalar_local); CHKERRQ(ierr);
\end{verbatim}

is followed with a \texttt{VecLoad} petsc stops after a strange error message.

30. seems not possible to get an off-line version of all user manuals of petsc?

31. seems not possible to do a search on the petsc website.

32. The PETSc Manual Pages and Examples sections such as:

- Vector Operations (Vec)
- Distributed Arrays (DA)

on page: \url{http://www-unix.mcs.anl.gov/petsc/petsc-2/documentation/index.html} do not mention all related methods. For instance: The method

\begin{verbatim}
VecPackScatter()
\end{verbatim}
is not mentioned under Vector Operations (Vec) but under Distributed Arrays (DA).

33. On the use of the VecPack functions. I thought it could sole be used with matrix free methods because it is not possible to create a matrix related to a VecPack constructed Vector (see section: comments on petsc). However, one can use a common matrix allocated with MatCreate for the Jacobian, and in FormJacobian fill it with MatSetValue/MatSetValues (see snes/ex3.c) – but NOT with MatSetValuesStencil().

34. On the use the block vector + matrix support. Problem: Block matrix support is not ensured to function with different petsc concepts/matrix representations. No explanation as to what does not function.

In order to see all methods Vec*, use: "Index of all manual pages" (double click the word Index) and look for V*.

6 References
1. C.R. Drago and A.M. Anile, An optimal control approach for an energy transport model in semiconductor design.

7 Numerical tests

7.1 How to the programs are tested (the ideas)

The codes are tested with: linear solution – to test whether solution is exact if it has to be; quadratic solution – to test whether solution is exact if it has to be; a non-polynomial solution – to test whether ||solution – approximation||max is of the right order $O(h^p)$.

7.2 Not conducted but possible to extra test

- Varying convection coefficients: Use: $\nabla \cdot (u \cdot b) = b \nabla u + u \nabla b$ Assume: (1) $\nabla_h$ is exact for degree 2 and (2) $u$ and $b$ both linear. Then $b \nabla u$ is discretized without error because $u \nabla b$ is and because $\nabla_h(u \cdot b)$ is (latter contains two terms with $\nabla_h$ operators applied to degree 2 polynomials).
7.3 Details

In general: do Several manners possible to run a compiled example with name ex1:

1. start mpd deamon just once(*): use: mpd &

2. use: mpiexec ./ex1 or

3. use: mpirun -n 1 ./ex1 or

4. use: mpirun -n 5 ./ex1 or

5. use: make runex1 or

6. use: make runex1_3

(*): Installation of petsc does step [0]. If mpd must run automatic after a next login, put the command in the .profile file.

- ex0c.: fill a non-sparse vector and print it
  - fill a sparse matrix and print it
  - explain smart/non-smart pitfalls in this file

  make ./ex0 # builds the program
  make runex0 # runs program with 1 processor
  make runex0_2 # runs program with 2 processors
  make clean

- ex1c.: make a non-uniform tensor grid and print it: ex1

  make ex1 # builds the program
  make runex1 # runs program with 1 processor
  make runex1_3 # runs program with 3 processors
  make runex1_3 | sort # same as make runex1_3, also sort the output
  mpiexec ./ex1 # same as make runex1: 1 process
  mpirun -n 1 ./ex1 # same as make runex1: 1 process
  mpirun -n 3 ./ex1 # same as make runex1_3: 3 processes
  mpirun -n 3 ./ex1 -n 5 -m 5 # same as previous, with options
  mpirun -n 3 ./ex1 -n 5 -m 5 | sort # same as previous, sorted on processor

  mpirun -n 2 ./ex6 -help | less # show all the command line options, look at the pages of options!

  make clean # deletes the executable (each about 50Megabytes)

Note: output from different processes can be mixed even inside a line
  if this happens, rerun or run with a smaller problem size

31
o ex1.c: load a non-sparse (da-) vector and
 save a non-sparse (da-) vector
 (to facilitate communication with outside -- for instance the state solver)

 mpirun -n 1 ./ex1
 mpirun -n 3 ./ex1

 o ex2.c: setup/solve the equations $F(U) = S$ (for given $S$) on a uniform grid

 make ex2
 make runex2
 mpirun -n 3 ./ex2 # use default GMRES
 mpirun -n 3 ./ex2 -snes_view # print solver info
 mpirun -n 3 ./ex2 -snes_view -snes_monitor # also print $F(U)$ residual norms
 mpirun -n 3 ./ex2 -ksp_type bicg -snes_view -snes_monitor # use BICG instead of GMRES
 mpirun -n 3 ./ex2 -ksp_type bicg -snes_view -snes_monitor -mat_view # also print $dF(U)$
 make clean # deletes the executable (each about 50Megabytes)

 for more options, see tutorial.pdf by Matthew Knepley, petsc developer, p31

... 

8 To do to finish the adjoint problem

First, adjusts the dimension:

1. Translate the 2d example (exercise with the highest number) into 1d;

The adjoint operator itself:

1. Finish the adjoint operator in FormFunction2/FormJacobian2 (run with -help to see
   which operator):

   (a) The first scalar equation: define it in case nb_operator = 4 in FormFunction1/FormJacobian1;

   (b) The second scalar equation: define it in case nb_operator = 5 in FormFunction1/FormJacobian1;

   (c) The vector equations: define it in case nb_operator = 2 in FormFunction2/FormJacobian2;

2. Ensure that the adjoint operator is finished with a known solution. Follow the stan-
   dard technique from FormFunction*:
... 

// - a_{11}\laplace s_1
_F[j][i].u -= (- a_{11}*s1_{xx} - a_{11}*s1_{yy});
_F[j][i].u += a_{11} * (- h_{right_x}*U[j][i-1].u + (h_{left_x} + h_{right_x})*U[j][i].u - h_{left_x}*U[j][i+1].u)/(h_{mean_cubed_x});
_F[j][i].u += a_{11} * (- h_{right_y}*U[j-1][i].u + (h_{left_y} + h_{right_y})*U[j][i].u - h_{left_y}*U[j+1][i].u)/(h_{mean_cubed_y});
...

where the line \_F[j][i].u -= ( - a_{11}*s1_{xx} - a_{11}*s1_{yy}); ensures that s1 is the solution.

3. Make sure to test the program (see tests with ex15.c):

set -solution_1 and -solution_2 and -solution_vector to 1 (linear) and run with n=m=5, 9, 17, 33, 65 must result in an inf error approx 0
set -solution_1 and -solution_2 and -solution_vector to 2 (quadratic) and run with n=m=5, 9, 17, 33, 65 must result in an inf error which decreases with a factor 4 (O(h^2)) or factor 2 (O(h))

4. If the tests show that the operator is ok then make the example physical: zero right hand sides:

... 

// - a_{11}\laplace s_1
_F[j][i].u -= 0;
_F[j][i].u += a_{11} * (- h_{right_x}*U[j][i-1].u + (h_{left_x} + h_{right_x})*U[j][i].u - h_{left_x}*U[j][i+1].u)/(h_{mean_cubed_x});
_F[j][i].u += a_{11} * (- h_{right_y}*U[j-1][i].u + (h_{left_y} + h_{right_y})*U[j][i].u - h_{left_y}*U[j+1][i].u)/(h_{mean_cubed_y});
...

Define this new operator as a new operator:

(a) The first scalar equation: define it in case nb_operator = 6 in FormFunction1/FormJacobian1;
(b) The second scalar equation: define it in case nb_operator = 7 in FormFunction1/FormJacobian1;
(c) The vector equations: define it in case nb_operator = 3 in FormFunction2/FormJacobian2;

5. Note that now the test against the exact solution make no sense, so this output should not be looked at;

6. Adapt the BCs for the adjoint operator, use _input_scalar[j][i] which contains the current I_g for the scalar problem at grid point (C[j][i].x, C[j][i].y)

7. Adapt the BCs for the adjoint operator, use _input_component[j][i].u which contains the solution \phi of the scalar problem at grid point (C[j][i].x, C[j][i].y)

8. Use the new coefficients: Instead of a_{11} use a_{11}(_input_vector, C, j, i)
The problem’s coefficients:

1. Replace the global scalars such as PetscScalar a\_11 etc. with global functions

   PetscScalar a\_11(FieldInput **F, DACoor2d **x, PetscInt j, PetscInt i) { ... } (2 dimensions)
   PetscScalar a\_11(FieldInput *F, DACoor1d *x, PetscInt i) { ... } (1 dimension)

2. Inside the global functions ... use the definitions from section 2.7 – or while in transition/test phase, use the old constant numerical values.

Use the input \( I_g \) and \( [w_1, w_2, V] \):

1. Use the real \( I_g \), defined as a function in file scalar.bin; and
2. the real \( C \), defined as a function in file scalar2.bin; and
3. the real \( [w_1, w_2, V] \) defined as a vector function in file vector.bin:
4. To the latter end delete all between the lines (twice):

   // START: TO BE DELETED
   ...
   // END: TO BE DELETED

5. Use the ascii to petsc-binary conversion in order to create scalar.bin and vector.bin from ascii input

9 The future

Different examples are:

- \texttt{ex6.c} scalar convection diffusion equation;
- \texttt{ex9.c} vector convection diffusion equation – but with simple vector operators
- \texttt{ex13.c} scalar and vector convection diffusion equations – but with more complex vector operators
- \texttt{ex15.c} scalar and vector convection diffusion equations + communication (WITH INCORRECT NEUMANN OPERATOR!)

Note: \texttt{ex10, ex11, ex12 and ex15} HAVE INCORRECT NEUMANN OPERATOR!
9.1 Improvements of the code

- Use flex dimension PetscScalar[3] instead of .u, .v, .w ⇒ but 1 FormFunction/Jacobian needed;
- ... but there are still 3 das needed (1-dof ... 3-dof);
- MAAK HET ZO DAT ELKE OPERATOR ZIJN EIGEN BCS HEEFT EN ADD THE BCS IN THE "HELP"-STRING; left: neumann, elswise dirichlet
- diagonal, off_diagonal is an option for scalar operators
- Ensure that each operator (each case in the operator switch) has its own IJ loop;
- Ensure that scalar cases’ DOF loop is around the IJ loop;
- Make options of the (diffusion, convection and source) coefficients ⇒ less operators required
- For the uniform grid, specify:
  1. Type and order of stencils for s’ and s” etc., for both INTERIOR and BOUNDARY stencils
  2. FROM THIS DERIVE THE WIDTH OF THE STENCIL!
- Print matrix in matlab sparse an option (OR USE -mat_view_matlab) (put into ”help”-string);
- Put the correct -neumann 2 code into ex15.c
- Make example ex15.c into ex16.c with a switch for scalar and/or vector code – so one example can do scalar and/or vector. Get rid of the communication and the load and save.
- Observe that the -neumann 2 stencil in fact has width 2 !!!!!!!! Hence it is possible but depends on the processor dofs partition whether this will hurt.
- Make improvements in Jacobian fill out: preserve possible symmetry
- Scale the Jacobian’s rows
- Abbreviate rows (r) and columns (c), delete nb_rows and nb_columns
- duplicate preload stencil into matrix stencil
- preload zero value for all of and i, j, k
- constant coefficients:
- duplicate matrix stencil values into matrix (not add);
- increase i or j or k for all column numbers in the stencil and for the row number

- non-constant coefficients:
  - first zero the matrix stencil's values
  - scale-add preloaded stencil values onto the matrix stencil values
  - ...
  - scale-add preloaded stencil values onto the matrix stencil values
  - duplicate matrix stencil values into matrix (not add)
  - increase i or j or k for all column numbers in the stencil and for the row number

- amount of dimensions is coded in the stencils + in the DGetCorners (i, j, k loop)
  - external loop over all grid points k, j, l is cheap if k=1,1,1, so no problem
  - internal loops can also be fast, if the stencils are precomputed – which is for sure possible with constant coefficients and also should be possible with non-constant coefficients! reason for efficient internal loops: if the stencil is precomputed, just one loop over the amount of column entries is required, instead of a triple loop to first fill out the stencil – hence the stencils should be precomputed BEFORE the interior loop

- what about an operator $\Delta x + \Delta y$ in combination with a 3-d problem?

- operator specification? (what is $\nabla x$ in 1 dimension?)

### 9.2 Optimizations

See also section 9.3.

- Ensure compilation is with `BOPT=0` instead of `BOPT=g` (petsc manual, p142): example:

  ```bash
tt make BOPT=0 ex1.
  ```

- Scale the equations with a factor $h^2$ such that $F(U^{(0)})$ becomes smaller and `snes` behaves better – it uses the size of the residual to determine how exact the linear systems must be solved and if the size of the residual is large the linear systems are solved rather inexact. This leads to problems with linear systems if the tolerance for the non-linear case and linear case are not chosen right (petsc does not choose them right).

- Ensure the correct factor `-pc_ilu_fill` (see petsc manual page 143 for details)
• Ensure the correct reorder strategy for the ILU(0): -pc_lu_ordering_type rcm or otherwise, see petsc manual page 165

• tensor functions do not take the domain into account (assume $[0,1]^2$)

• use *flag = SAME_NONZERO_PATTERN; as in snes/ex14.c (for nonlinear problems) also use in this case: ierr = MatSetOption(jac,MAT_NEW_NONZERO_LOCATION_ERR);

• use application tuned preconditioners

• invoke multi-grid, should at least work for uniform grids

• Do not use DA’s weak tensor grid but a local $x_{ii}, \ldots, z_{ii}$ vector for the tensor grid ⇒ uses less memory

• geometric input would be nice, now the partition and related conditions of the boundary is hardcoded into the Form*()

• Use option MAT_COLUMNS_SORTED (not applicable when we use MatSetValuesStencil())

• use preallocation for the sparse matrices used ⇒ more speed (done automatic for DAMatrices, so not needed)

Possible optimization of the Jacobian:

• The Jacobian matrix is filled in in the standard petsc manner (see snes/ex14.c etc). This leads to a non-symmetric and potentially non-positive definite Jacobian (due to the manner in which dirichlet points are treated). It is possible to alter the Dirichlet points different and to get a symmetric matrix (if no convection is present).

• The standard finite difference approach in the examples introduces differently scaled equations and hence large residuals $F(U^{(0)})$. Better is to scale the equations:
  1. factor $O(1)$ for dirichlet dofs
  2. factor $O(h)$ for neumann dofs
  3. factor $O(h^2)$ for pde related dofs

• The stencils choosen in the examples are such that they do not destruct diagonal dominance or positive definiteness when possible. See comments in FormJacobian at several places. When adding examples, whatch out that the diagonal dominance etc., is not ruined with the addition of the wrong stencils.
9.3 The research wish list

See also section 9.2.

1. Debug neumann == 2 case (stencil is not correct, it seems)

2. Test whether \( h_i^3 h_{i+1} - h_{i+1}^3 h_i = 0 \) for our tensor function \texttt{tensor\_bubble()} on a grid of \( h_i = [x_i - x_{i-1}] \) with initial \( x_i = i/N(i = 0, \ldots, N) \) with \( N \) the amount of grid points.

3. determine which non-linear solver is called with snes and what its stop criterion is (seems impossible to set a fixed stop criterion to the linear solver inside a non-linear solver).

4. setup/solve a laplace problem (dirichlet bcs) with superlu

5. determine whether micro/macro-block matrix switch support exists for multiple component systems

6. use parallel multigrid (possible if we use \texttt{DAGet(Ghosted)Coordinates()})

7. use finite elements rather than differences (use libmesh (c++))

8. Couple the statesolver to the adjoint solver
   - Use a unix/python script to alternate between adjoint and state solvers Translation to the variables required for the adjoint problem are best done in parallel (petsc) – but no time?
   - Use a c/c++ program to couple both – but execute the serial state solver just at the processor with rank 0.