Numerical Schemes -2-

Thematic School Math-Info-HPC

Thierry Dumont

Institut Camille Jordan, Lyon.

May 10, 2016

Sketch of the talk

Some promising techniques

- Avoid solution of linear systems with stabilized explicit Runge–Kutta methods
- Discontinuous Galerkin methods

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \ : \ [t_0, +\infty[\times \mathbb{R}^m \to \mathbb{R}^m.$$

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \ : \ [t_0, +\infty[\times \mathbb{R}^m \to \mathbb{R}^m.$$

Let u_k be the approximation of u at time $k\ \delta t$.

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \; : \; [t_0, +\infty[\times \mathbb{R}^m \to \mathbb{R}^m.$$

Let u_k be the approximation of u at time $k\ \delta t$. u_{k+1}?

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \ : \ [t_0, +\infty[\times\mathbb{R}^m \to \mathbb{R}^m.$$

Let u_k be the approximation of u at time $k \ \delta t$. u_{k+1}?

$$U_i \quad = \quad u_k + \delta t \sum_{j=1}^{s} a_{ij} F\left(t_0 + c_j \delta t, U_j\right), \quad i = 1, \ldots, s,$$

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \ : \ [t_0, +\infty[\times \mathbb{R}^m \to \mathbb{R}^m.$$

Let u_k be the approximation of u at time $k\ \delta t$. u_{k+1}?

$$\begin{aligned} U_i &= u_k + \delta t \sum_{j=1}^{s} a_{ij} F\left(t_0 + c_j \delta t, U_j\right), \quad i = 1, \ldots, s, \\ u_{k+1} &= u_k + \delta t \sum_{j=1}^{s} b_j F\left(t_0 + c_j \delta t, U_j\right), \end{aligned}$$

Runge–Kutta methods

$$\frac{du}{dt} = F(t,u), \quad u(t_0) = u_0, \quad F \; : \; [t_0, +\infty[\times \mathbb{R}^m \to \mathbb{R}^m.$$

Let u_k be the approximation of u at time $k \ \delta t$. u_{k+1}?

$$\begin{aligned} U_i &= u_k + \delta t \sum_{j=1}^{s} a_{ij} F\left(t_0 + c_j \delta t, U_j\right), \quad i = 1, \dots, s, \\ u_{k+1} &= u_k + \delta t \sum_{j=1}^{s} b_j F\left(t_0 + c_j \delta t, U_j\right), \end{aligned}$$

Far from the old RK4 method!

- explicit methods,
- implicit methods.

Runge-Kutta methods: Butcher array

$$\begin{array}{c|ccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s-1} & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s-1} & a_{2s} \\
\vdots & \vdots & & \ddots & & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss-1} & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Diagonal and upper diagonal $== 0 <=>$ explicit method.

Runge-Kutta methods: Butcher array

$$\begin{array}{c|ccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s-1} & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s-1} & a_{2s} \\
\vdots & \vdots & & \ddots & & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss-1} & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Diagonal and upper diagonal $== 0 <=>$ explicit method.

Implicit: need to solve an algebraic system of size $m \times s$ (use simplified Newton iterations), and a linear system if the problem is linear.

Runge-Kutta methods: diagonally implicit methods

$$\begin{array}{c|ccccc}
c_1 & \gamma & 0 & \cdots & \cdots & 0 \\
c_2 & a_{21} & \gamma & \cdots & 0 & 0 \\
\vdots & \vdots & & \ddots & & \vdots \\
c_{s-1} & a_{s-1,1} & a_{s-1,2} & \cdots & \gamma & 0 \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \gamma \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Solve sequentially s systems. For a linear system of ODEs, solve s linear systems, **with the same matrix**.

Runge-Kutta methods: diagonally implicit methods

$$
\begin{array}{c|ccccc}
c_1 & \gamma & 0 & \cdots & \cdots & 0 \\
c_2 & a_{21} & \gamma & \cdots & 0 & 0 \\
\vdots & \vdots & & \ddots & & \vdots \\
c_{s-1} & a_{s-1,1} & a_{s-1,2} & \cdots & \gamma & 0 \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \gamma \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

Solve sequentially s systems. For a linear system of ODEs, solve s linear systems, **with the same matrix**.

- Fully implicit RK methods cannot be used for solving PDEs.
- Diagonally implicit RK methods are ok, but with a lot of linear algebra (for linear PDEs).

Runge–Kutta methods: stability

Look at a linear problem: $dy/dt = \lambda y$, with $\lambda \in \mathbb{C}$.

Runge–Kutta methods: stability

Look at a linear problem: $dy/dt = \lambda y$, with $\lambda \in \mathbb{C}$.
Then set $z = \delta t \lambda$.
We have:

- With an explicit method: $x_{k+1} = P(z) x_k$.
- With an implicit method: $x_{k+1} = Q(z) x_k$ (Padé approximant of $\exp$).

Runge–Kutta methods: stability

Look at a linear problem: $dy/dt = \lambda y$, with $\lambda \in \mathbb{C}$.
Then set $z = \delta t \lambda$.
We have:

- With an explicit method: $x_{k+1} = P(z) x_k$.
- With an implicit method: $x_{k+1} = Q(z) x_k$ (Padé approximant of $\exp$).

Stability domain: $\mathcal{S} = \{z \in \mathbb{C} |\ |x_{k+1}| \leq |x_k|\}$.

Runge–Kutta methods: stability

Look at a linear problem: $dy/dt = \lambda y$, with $\lambda \in \mathbb{C}$.
Then set $z = \delta t \lambda$.
We have:

- With an explicit method: $x_{k+1} = P(z) x_k$.
- With an implicit method: $x_{k+1} = Q(z) x_k$ (Padé approximant of $\exp$).

Stability domain: $\mathcal{S} = \{z \in \mathbb{C} | \; |x_{k+1}| \leq |x_k|\}$.
If method is:

- explicit: $\mathcal{S}$ is bounded.
- implicit: $\mathcal{S}$ possibly unbounded in some direction.
  If $\{x \in \mathbb{C}, \mathrm{Re}(x) < 0\} \subset S$, method is said A-stable.

Runge–Kutta methods: stability

Look at a linear problem: $dy/dt = \lambda y$, with $\lambda \in \mathbb{C}$.
Then set $z = \delta t \lambda$.
We have:

- With an explicit method: $x_{k+1} = P(z) x_k$.
- With an implicit method: $x_{k+1} = Q(z) x_k$ (Padé approximant of $\exp$).

Stability domain: $\mathcal{S} = \{z \in \mathbb{C} |\ |x_{k+1}| \leq |x_k|\}$.
If method is:

- explicit: $\mathcal{S}$ is bounded.
- implicit: $\mathcal{S}$ possibly unbounded in some direction.
  If $\{x \in \mathbb{C}, \mathrm{Re}(x) < 0\} \subset S$, method is said A-stable.

Which value of δt is allowed?

- explicit: $\delta t \simeq$ smallest time scales ($\delta t \leq 1/|\lambda_{\max}|$ for linear systems of ODEs).
- A-stable: δt only limited by precision.

Runge–Kutta methods: order

Definition (order of an ODE solver)

Consider $dy/dt = f(y)$ *starting from* y_0 *at time* $t = 0$.
Apply the solver with a time step $\delta t =\!> y_1$ *and compare* y_1 *and the exact solution* $y(\delta t)$.

Runge–Kutta methods: order

Definition (order of an ODE solver)

Consider $dy/dt = f(y)$ starting from y_0 at time $t = 0$.
Apply the solver with a time step $\delta t => y_1$ and compare y_1 and the exact solution $y(\delta t)$.
Method is of order p iff the first p coefficients of the Taylor expansions of y_1 and $y(\delta t)$ as functions of δt are equal.

Runge–Kutta methods: order

Definition (order of an ODE solver)

Consider $dy/dt = f(y)$ starting from y_0 at time $t = 0$.
Apply the solver with a time step $\delta t => y_1$ and compare y_1 and the exact solution $y(\delta t)$.
Method is of order p iff the first p coefficients of the Taylor expansions of y_1 and $y(\delta t)$ as functions of δt are equal.

Note: computing the Taylor expansions is not easy: special tools invented by J. Butcher (rooted trees).

Runge–Kutta methods: order

Definition (order of an ODE solver)

Consider $dy/dt = f(y)$ starting from y_0 at time $t = 0$.
Apply the solver with a time step $\delta t => y_1$ and compare y_1 and the exact solution $y(\delta t)$.
Method is of order p iff the first p coefficients of the Taylor expansions of y_1 and $y(\delta t)$ as functions of δt are equal.

Note: computing the Taylor expansions is not easy: special tools invented by J. Butcher (rooted trees).

Some general results (s is the number of stages of the method):

- Implicit methods can have order up to $2s$ (Gaussian method of Kuntzman and Butcher).
- explicit methods with s stages cannot be of order $> s$.

Explicit, but stabilized, Runge–Kutta methods

The idea (1): do not optimize the method for the order o, but for the size of the stability domain; we will have $s >> o$.

Explicit, but stabilized, Runge–Kutta methods

The idea (1): do not optimize the method for the order o, but for the size of the stability domain; we will have $s >> o$.

For a linear problem $dy/dt = \lambda y$, applying one step $y_n \to y_{n+1}$ of the method can be seen as applying a polynomial of $z = \delta t.\lambda$ to y_n:
$y_{n+1} = P_s(z) y_n$ (degree of $P_s = s$).

Explicit, but stabilized, Runge–Kutta methods

The idea (1): do not optimize the method for the order o, but for the size of the stability domain; we will have $s >> o$.

For a linear problem $dy/dt = \lambda y$, applying one step $y_n \to y_{n+1}$ of the method can be seen as applying a polynomial of $z = \delta t.\lambda$ to y_n: $y_{n+1} = P_s(z) y_n$ (degree of $P_s = s$).

The idea (2): find the polynomial P_s which optimize the stability domain, and then (try to) build the RK method from it.

Explicit, but stabilized, Runge–Kutta methods

The idea (1): do not optimize the method for the order o, but for the size of the stability domain; we will have $s >> o$.

For a linear problem $dy/dt = \lambda y$, applying one step $y_n \rightarrow y_{n+1}$ of the method can be seen as applying a polynomial of $z = \delta t.\lambda$ to y_n:
$y_{n+1} = P_s(z) y_n$ (degree of $P_s = s$).

The idea (2): find the polynomial P_s which optimize the stability domain, and then (try to) build the RK method from it.

For order 1, P_s is the shifted sth Chebyshev polynomial.

Explicit, but stabilized, Runge–Kutta methods

The idea (1): do not optimize the method for the order o, but for the size of the stability domain; we will have $s >> o$.

For a linear problem $dy/dt = \lambda y$, applying one step $y_n \to y_{n+1}$ of the method can be seen as applying a polynomial of $z = \delta t.\lambda$ to y_n:
$y_{n+1} = P_s(z) y_n$ (degree of $P_s = s$).

The idea (2): find the polynomial P_s which optimize the stability domain, and then (try to) build the RK method from it.

For order 1, P_s is the shifted sth Chebyshev polynomial.
Obtaining an order > 1?

Explicit, but stabilized, Runge–Kutta methods

Abdulle and Medovikov: methods of order 2 and 4: Rock2 and Rock4.

Explicit, but stabilized, Runge–Kutta methods

Abdulle and Medovikov: methods of order 2 and 4: Rock2 and Rock4.

Not easy: use the fact that the set of RK methods is a group for the composition of functions (the Butcher group).

Explicit, but stabilized, Runge–Kutta methods

Abdulle and Medovikov: methods of order 2 and 4: Rock2 and Rock4.

Not easy: use the fact that the set of RK methods is a group for the composition of functions (the Butcher group).

- The Jacobian of the RHS must have eigenvalues near the real axis.
- s (the number of stages) vary from 5 to more than 100 (defined by the largest eigenvalue of the Jacobian of the RHS).
- You must have an estimation of the largest eigenvalue of the Jacobian.

Explicit, but stabilized, Runge–Kutta methods

Use them for the Heat equation.

$$\frac{du}{dt} = \varepsilon \Delta u.$$

In many cases, we have $\delta t\ \varepsilon$ *not large*.
Examples:

- Biological problems: ε is small.
- Reaction diffusion equations: systems of the form:

$$\frac{du_i}{dt} = \varepsilon_i \Delta u_i + f_i(u_1, \ldots, u_n), \quad i = 1, n.$$

  The fastest time scales are in the (chemical) reaction.

Explicit, but stabilized, Runge–Kutta methods

Use them for the Heat equation.

$$\frac{du}{dt} = \varepsilon \Delta u.$$

In many cases, we have $\delta t\ \varepsilon$ *not large*.
Examples:

- Biological problems: ε is small.
- Reaction diffusion equations: systems of the form:

$$\frac{du_i}{dt} = \varepsilon_i \Delta u_i + f_i(u_1, \ldots, u_n), \quad i = 1, n.$$

The fastest time scales are in the (chemical) reaction.

Implementation

For linear problem the method reduces to $U_{n+1} = P_s(A)U_n$. Use Horner rule to evaluate it.

Avoid any solution of linear systems.

Discontinuous Galerkin methods

- Mixed formulation of $\Delta u = f$:
  $\operatorname{div} \vec{\sigma} = f$
  $\vec{\sigma} = \overrightarrow{\operatorname{grad}}\, u.$
- Use of domain decomposition of Ω in disjoint parts $\Omega = \cup K_h$
- Use Green formula to write the mixed formulation on each K, performing some "integration by part".

DG Methods

Unknowns are σ_h and u_h.

$$\begin{aligned}\int_K \sigma_h.\tau dx = & -\int_K u_h \operatorname{div} \tau dx + & \int_{\partial K} \hat{u}_K \eta_K.\tau ds \quad \forall \tau \in \Sigma(K),\\ \int_K \sigma_h.\vec{\operatorname{grad}}\, v dw = & \int_K f v dx \quad + & \int_{\partial K} \hat{\sigma}_K.\eta_K v ds \quad \forall v \in P(K).\end{aligned}$$

DG Methods

Unknowns are σ_h and u_h.

$$\begin{aligned} \int_K \sigma_h.\tau dx &= -\int_K u_h \operatorname{div} \tau dx + \int_{\partial K} \hat{u}_K \eta_K.\tau ds \quad \forall \tau \in \Sigma(K), \\ \int_K \sigma_h. \vec{\operatorname{grad}}\, v dw &= \int_K f v dx + \int_{\partial K} \hat{\sigma}_K.\eta_K v ds \quad \forall v \in P(K). \end{aligned}$$

- Σ and P are generally polynoms.
- $\hat{\sigma}_K$ and $\hat{u}_K$ are numerical fluxes; that is to say well chosen approximations of the terms which appear when doing the integration by part (the problem must be well posed: penalisation terms must be added; all the art is here).
- $\hat{\sigma}$ is interesting in many applications (example: flows in porous media).

DG / IP Method

Many choices for the available fluxes are available. The Interior Penalty method is convenient: it has good numerical properties and the stencil generated is quite small.
Let K_1 and K_2 be 2 neighbor elements with a common edge e.

$$\begin{aligned}
\phi(x) \in \mathbb{R}^d \quad &: \quad \phi = \frac{1}{2}(\phi_1 + \phi_2) \qquad [\phi] = \phi_1 . \eta_1 + \phi_2 . \eta_2, \\
\phi(x) \in \mathbb{R} \quad &: \quad \phi = \frac{1}{2}(\phi_1 + \phi_2) \qquad [\phi] = \phi_1 \eta_1 + \phi_2 \eta_2.
\end{aligned}$$

Interior penalty method. Fluxes

$\hat{u} = u_h, \quad \hat{\sigma} = \vec{\operatorname{grad}}\, u_h - \eta_e h_e^{-1}[u_h].$

But one can eliminate σ_h:

Interior penalty method. Primal form

$$B_h(u_h, v) = \int_\Omega \vec{\operatorname{grad}}\, u_h . \vec{\operatorname{grad}}\, v dx - \int_\Gamma ([u_h].\vec{\operatorname{grad}}\, v + \vec{\operatorname{grad}}\, u_h.[v]) ds + \int_\Gamma \alpha[u_h].[v] ds.$$

with $\alpha = \eta_e h_e^{-1}$on each $e \in \mathcal{E}$.

Here: $\int_\Omega \dots dx = \sum_k \int_k \dots dx$.

Solve:

$$B_h(u_h, v) = \int_\Omega f v dx.$$

DG

On cartesian grids (cubes) implement the method using:

- Legendre basis:

$$Q_{i,j,k} = P_{i,j,k}(x,y,z) = p_i(x)\ p_j(y)\ p_k(z),$$

  with:

$$p_l(s) = L_l((2s-h)/h),\ l = 0, \text{degree}.$$

  (normalized to obtain an identity mass matrix).
- for degrees from 2 to 5 (thanks to SageMath software).

Best results for degree 3:

- I_a grows with the degree of polynomials.
- Computers like vectors of size divisible by 4.

DG: stencil for polynomials of degree 3

- On 3d cartesian grid, we get a 7 *matrices* stencil. Let $A_{i,j}$ be these matrices.
- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j} = PBP^{-1}$ or$A_{i,j} = PB^tP^{-1}$ where B is a 64×64 matrix made of 4×4 blocks on the diagonal.

DG: stencil for polynomials of degree 3

- On 3d cartesian grid, we get a 7 *matrices* stencil. Let $A_{i,j}$ be these matrices.
- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j} = PBP^{-1}$ or$A_{i,j} = PB^tP^{-1}$ where B is a 64×64 matrix made of 4×4 blocks on the diagonal.

I_a ?

- Flops:

$A_{i,j},\ i \neq j$	:	6	$\times$	512	$=$	3072	
$A_{i,i}$	:	1	$\times$	512	$=$	512	
Total	:					3584	flops.

DG: stencil for polynomials of degree 3

- On 3d cartesian grid, we get a 7 *matrices* stencil. Let $A_{i,j}$ be these matrices.
- $A_{i,i}$ is a 64×64 matrix with 4 non zero terms by line.
- If $i \neq j$, $A_{i,j} = PBP^{-1}$ or$A_{i,j} = PB^tP^{-1}$ where B is a 64×64 matrix made of 4×4 blocks on the diagonal.

I_a ?

- Flops:

$A_{i,j},\ i \neq j$	:	6	$\times$	512	$=$	3072	
$A_{i,i}$	:	1	$\times$	512	$=$	512	
Total	:					3584	flops.

- Memory bandwidth: $8 \times 64 \ = 512$ (double).

So, $I_a = 7$ without any reuse of data.

DG: stencil for polynomials of degree 3

$I_a = 7$.

- Peak theoretical performance:
  $7 \times 8.73 = 61.2$ Gigaflops/second.

DG: stencil for polynomials of degree 3

$I_a = 7$.

- Peak theoretical performance:
  $7 \times 8.73 = 61.2$ Gigaflops/second.
- Measured (Rock4, method using Horner scheme):
  $\partial_t u = \Delta U$: 67 Gigaflops/second.
  $\partial_t u = \Delta U + f(x)$: 66 Gigaflops/second.

Some data is reused.

The Poisson equation

Conjugate Gradient and Polynomial Preconditioning.

Chebyshev preconditioning:

Find $s \in \mathbb{P}_k$ which minimizes:

$$\max_{\lambda \in [a,b]} |1 - \lambda s(\lambda)|.$$

Solution is a shifted and scaled Chebyshev polynomial.

To solve $Ax = B$, use $M^{-1} = s(A)$ as preconditionner.

The Poisson equation

Conjugate Gradient and Polynomial Preconditioning.

Chebyshev preconditioning:

Find $s \in \mathbb{P}_k$ which minimizes:

$$\max_{\lambda \in [a,b]} |1 - \lambda s(\lambda)|.$$

Solution is a shifted and scaled Chebyshev polynomial.

To solve $Ax = B$, use $M^{-1} = s(A)$ as preconditionner.

Evaluation using the 3 terms recurrence formula.
See results of W. Vanroose:
`http://calcul.math.cnrs.fr/IMG/pdf/poisson_vanroose.pdf`

Conjugate Gradient Preonditioned

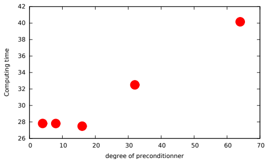
```
$r_0$:  $= b - Ax_0; u_0 = M^{-1}r_0; p_0 = u_0;$
for  $i = 0, ....$ do :
        $s := Ap_i$
        $\alpha :=< r_i, u_i > / < s, p_i >$
        $x_{i+1} := x_i + \alpha p_i$
        $r_{i+1} := r_i - \alpha s$
        $u_{i+1} := M^{-1} r_{i+1}$
        $\beta :=< r_{i+1}, u_{i+1} > / < r_i, u_i >$
        $p_{i+1} := u_{i+1} + \beta p_i$
```

GCP

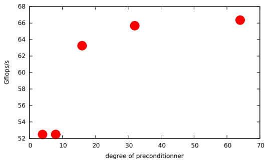
```
$r_0$:  $= b - Ax_0; u_0 = M^{-1}r_0; p_0 = u_0;$
for     $i = 0, .... \text{ do }:$
        $s := Ap_i$
        $\alpha := < r_i, u_i > / < s, p_i >$
        $r_{i+1} := r_i - \alpha s$
        $u_{i+1} := M^{-1} r_{i+1}$
        $\beta := < r_{i+1}, u_{i+1} > / < r_i, u_i >$
        $x_{i+1} := x_i + \alpha p_i$
        $p_{i+1} := u_{i+1} + \beta p_i$
```

GCP: results

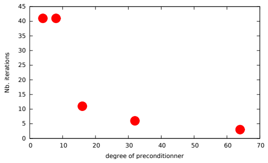
Grid 128^3 elements (512^3 unknowns), $-\Delta u = f$.



Computing time, best: degree $= 16$



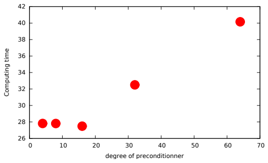
Gflops/s.



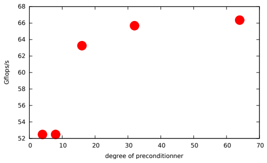
Nb. iterations

GCP: results

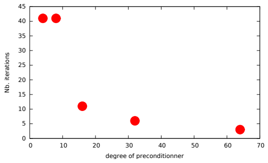
Grid 128^3 elements (512^3 unknowns), $-\Delta u = f$.



Computing time, best: degree $= 16$

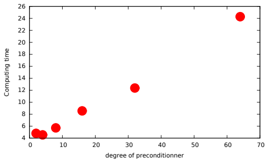


Gflops/s.

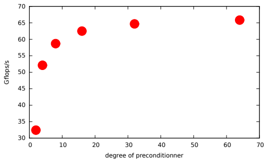


Nb. iterations

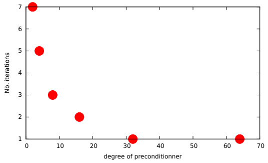
Grid 128^3 elements (512^3 unknowns), $-\Delta u + 0.01u = f$.



Computing time, best: degree $= 4$



Gflops/s.



Nb. iterations

Implementation, tuning

- Explore many possibilities with Python generated C++ and Jinja template engine.
- Intel compiler options: -O2 -restrict -std=c++11 -xHOST -no-prec-div
- VTune.

BLAS performances (Intel) on Sandy-Bridge, 8×2 cores, doubles.