RK-Opt User Manual

Release 0.2

David I. Ketcheson, Matteo Parsani, and Aron Ahmadia

April 07, 2013

RK-opt is a MATLAB package for designing Runge-Kutta (RK) methods and stability polynomials. Supported objective functions include the principal error norm and the SSP coefficient. Supported constraints include stability polynomial coefficients, low-storage formulations, and structural constraints (explicit, diagonally implicit, etc.) RK-opt uses MATLAB's optimization toolbox, in particular *fmincon* and *linprog*.

MATLAB's global optimization toolbox function *Multistart* can be used to exploit the benefits of parallel search on multicore machines.

The RK-Opt package consists of the following packages:

- **RK-coeff-opt:** Find optimal Runge-Kutta method coefficients for a prescribed order of accuracy and number of stages. The objective function can be chosen as either the **SSP coefficient** or the **leading truncation error coefficient**. The method may be constrained to have a **low-storage implementation** and/or a prescribed **stability polynomial**. Implicit and diagonally implicit methods can also be optimized.
- **am_rad-opt:** Find stability functions with optimal radius of absolute monotonicity. This includes codes for optimizing stability functions of multistep, multistage methods and even methods with downwinding. The optimization of rational functions is experimental.
- **polyopt:** Given a spectrum (typically corresponding to a spatial semi-discretization of a PDE), find an optimal stability polynomial in terms of its coefficients. These polynomial coefficients can then be used as input to **RK-coeff-opt** to find a corresponding Runge-Kutta method.
- **RKtools:** Some general utilities for analyzing Runge-Kutta methods.

If you use RK-Opt in published work, please cite this manual and/or relevant papers from the bibliography below.

CHAPTER

ONE

RK-OPT

1.1 Automated design of Runge-Kutta methods

An s-stage Runge-Kutta method has roughly s^2 coefficients (roughly $s^2/2$ for explicit methods), which can be chosen so as to provide high accuracy, stability, or other properties. Historically, most interest in Runge-Kutta methods has focused on methods using the minimum number of stages for a given order of accuracy. However, in the past few decades there has been increasing recognition that using *extra* stages can be worthwhile in order to improve other method properties. Some areas where this is particularly useful are in the enhancement of linear and nonlinear stability properties, the reduction of storage requirements, and the design of embedded pairs. Methods with dozens or even hundreds of stages are not unheard of.

At the same time, most existing Runge-Kutta methods have been designed by hand, by researchers laboriously solving the order conditions. When using extra stages, the number of available parameters makes the selection of a near-optimal choice by hand impossible, and one resorts to computational optimization. This leads to a different paradigm of numerical method design, in which we use sophisticated numerical (optimization) algorithms to design sophisticated numerical (integration) algorithms. It can be expected that this trend will accelerate in the future, and perhaps one day simple manually-constructed algorithms will be the exception.

RK-Opt contains a set of tools for designing Runge-Kutta methods in this paradigm. It has been constructed mostly in the direct line of our research, but we have made some effort to help others easily understand and use it. We hope that you find it useful, and that you will contribute any enhancements you may develop back to the project by sending us a pull request on Github.

1.2 Installation

This section describes how to obtain RK-opt and test that it is working correctly.

1.2.1 Dependencies

- MATLAB 7.X or greater
- MATLAB optimization toolbox

1.2.2 Optional dependencies

- MATLAB Global Optimization toolbox (for multicore search)
- xUnit test framework (for testing your installation); available for free from http://www.mathworks.com/matlabcentral/fileexchange/22846. xUnit requires MATLAB R2008a or later.

1.2.3 Obtaining RK-opt

- Download: https://github.com/ketch/RK-opt/downloads.
- Or clone:

```
$ git clone https://github.com/ketch/RK-opt.git
```

After unzipping/cloning, add the subdirectory `RK-opt/RKtools` to your MATLAB path.

1.3 Testing your installation

You can test your RK-opt installation with xUnit.

1.3.1 Installing xUnit

The MATLAB xUnit test framework can be downloaded for free at http://www.mathworks.com/matlabcentral/fileexchange/22846 (after the link, click on the button in the upper right corner). An easy way to install xUnit without setting any environment variables is to add the following lines to your `startup.m` file:

```
addpath /path/to/matlab/xunit/matlab_xunit
addpath /path/to/matlab/xunit/matlab_xunit/xunit
addpath /path/to/RK-opt/RK-coeff-opt
```

1.3.2 Running the tests

To run the tests, do the following in MATLAB:

```
>> cd /path/to/RK/Opt/general/test
>> runtests
```

If everything is set up correctly, this will run several tests, and inform you that the tests passed. At present the tests are not very extensive.

1.4 About RK-opt

RK-opt has been developed by David Ketcheson (primary developer and maintainer), Matteo Parsani, and Aron Ahmadia. It is released under a modified BSD License.

1.5 Roadmap

The most significant planned update to RK-opt is the addition of Python routines for the RK-coeff-opt package. These will use OpenOpt and in particular Ipopt.

Additionally, we have bits of code for the following that will be merged in to the main package:

- Optimization of coefficients for Runge-Kutta schemes with downwinding

1.6 Citing Us

If you use RK-Opt for published work, please consider citing this manual and any relevant works from our bibliography. Also, please do let us know if you are using this software so we can add your work to our Applications section.

CHAPTER

TWO

REFERENCE

This section contains a compilation of the documentation of each function, organized by subpackage.

2.1 RK-coeff-opt

This subpackage contains routines for finding optimal Runge-Kutta method coefficents, given a prescribed order of accuracy, number of stages, and an objective function. Constraints on the stability polynomial (possibly obtained using **polyopt** or **am_radius-opt**) can optionally be provided.

Contents

2.1.1 check_RK_order

```
function p = check_RK_order(A,b,c)
```

Determines order of a RK method, up to sixth order.

For an s-stage method, input A should be a $s \times s$ matrix; b and c should be column vectors of length s.

2.1.2 errcoeff

```
function D = errcoeff(A,b,c,p)
```

Inputs:

- A, b, c – Butcher tableau
- p – order of accuracy of the method

Computes the norm of the vector of truncation error coefficients for the terms of order $p+1$: (elementary weight - 1/(density of the tree)/(symmetry of the tree)

For now we just use Butcher's approach. We could alternatively use Albrecht's.

2.1.3 linear_constraints

```
function [Aeq,beq,lb,ub] = linear_constraints(s,class,objective,k)
```

This sets up:

- The linear constraints, corresponding to the consistency conditions $\sum_j b_j = 1$ and $\sum_j a_{ij} = c_j$.
- The upper and lower bounds on the unknowns. These are chosen somewhat arbitrarily, but usually aren't important as long as they're not too restrictive.

2.1.4 nonlinear_constraints

```
function [con,coneq]=nonlinear_constraints(x,class,s,p,objective,poly_coeff_ind,poly_coeff_val,k)
```

Impose nonlinear constraints:

- if objective = 'ssp' : both order conditions and absolute monotonicity conditions
- if objective = 'acc' : order conditions

The input arguments are:

- x: vector of the decision variables. See unpack_rk.m for details about the order in which they are stored.
- *class*: class of method to search ('erk' = explicit RK; 'irk' = implicit RK; 'dirk' = diagonally implicit RK; 'sdirk' = singly diagonally implicit RK; '2S', '3S', '2S*', '3S*' = low-storage formulations).
- s:number of stages.
- p: order of the RK scheme.
- *objective*: objective function ('ssp' = maximize SSP coefficient; 'acc' = minimize leading truncation error coefficient).
- *poly_coeff_ind*: index of the polynomial coefficients (β_j) for $j > p$.
- *poly_coeff_val*: values of the polynomial coefficients (β_j) for $j > p$ (tall-tree elementary weights).

The outputs are:

- *con*: inequality constraints, i.e. absolute monotonicity conditions if objective = 'ssp' or nothing if objective = 'acc'
- *coneq*: order conditions plus stability function coefficients constraints (tall-tree elementary weights)

Two forms of the order conditions are implemented: one based on **Butcher's approach**, and one based on **Albrecht's approach**. One or the other may lead to a more tractable optimization problem in some cases, but this has not been explored carefully. The Albrecht order conditions are implemented up to order 9, assuming a certain stage order, while the Butcher order conditions are implemented up to order 9 but do not assume anything about the stage order. Albrecht's approach is used by default.

2.1.5 oc_albrecht

```
function coneq=oc_albrecht(A,b,c,p)
```

Order conditions for SSP RK Methods.

This version is based on Albrecht's approach

2.1.6 oc_butcher

```
function coneq=oc_butcher(A,b,c,p)
```

Order conditions for RKMs. This version is based on Butcher's approach.

Assumes $p > 1$.

2.1.7 oc_ksrk

```
function coneq= oc_ksrk(A,b,D,theta,p)
```

Order conditions for multistep-RK methods.

2.1.8 order_conditions

```
function tau = order_conditions(x,class,s,p,Aeq,beq)
```

This is just a small wrapper, used when solveorderconditions=1.

2.1.9 rk_obj

```
function [r,g]=rk_obj(x,class,s,p,objective)
```

Objective function for RK optimization.

The meaning of the input arguments is as follow:

- x: vector of the unknowns.
- class: class of method to search ('erk' = explicit RK; 'irk' = implicit RK; 'dirk' = diagonally implicit RK; 'sdirk' = singly diagonally implicit RK; '2S', '3S', '2S*', '3S*' = low-storage formulations).
- s:number of stages.
- p: order of the RK scheme.
- objective: objective function ('ssp' = maximize SSP coefficient; 'acc' = minimize leading truncation error coefficient).

The meaning of the output arguments is as follow:

- r: it is a scalar containing the radius of absolute monotonicity if objective = 'ssp' or the value of the leading truncation error coefficient if objective = 'acc'.
- g: it is a vector and contains the gradient of the objective function respect to the unknowns. It is an array with all zero elements except for the last component which is equal to one if objective = 'ssp' or it is an empty array if objective = 'acc'.

2.1.10 rk_opt

```
function rk = rk_opt(s,p,class,objective,varargin)
```

Find optimal RK and multistep RK methods. The meaning of the arguments is as follows:

- s number of stages.
- k number of steps (1 for RK methods)
- p order of the Runge-Kutta (RK) scheme.
- class: class of method to search. Available classes:
  - 'erk' : Explicit Runge-Kutta methods
  - 'irk' : Implicit Runge-Kutta methods
  - 'dirk' : Diagonally implicit Runge-Kutta methods
  - 'sdirk' : Singly diagonally implicit Runge-Kutta methods
  - '2S', etc. : Low-storage explicit methods; see *Ketcheson, "Runge-Kutta methods with minimum storage implementations". J. Comput. Phys. 229(5):1763 - 1773, 2010*)
  - 'emsrk1/2' : Explicit multistep-Runge-Kutta methods
  - 'imsrk1/2' : Implicit multistep-Runge-Kutta methods
  - 'dimsrk1/2' : Diagonally implicit multistep-Runge-Kutta methods
- objective: objective function ('ssp' = maximize SSP coefficient; 'acc' = minimize leading truncation error coefficient) Accuracy optimization is not currently supported for multistep RK methods
- poly_coeff_ind: index of the polynomial coefficients to constrain (β_j) for $j > p$ (j denotes the index of the stage). The default value is an empty array. Note that one should not include any indices $i \leq p$, since those are determined by the order conditions.
- poly_coeff_val: constrained values of the polynomial coefficients (β_j) for $j > p$ (tall-tree elementary weights). The default value is an empty array.
- startvec: vector of the initial guess ('random' = random approach; 'smart' = smart approach; alternatively, the user can provide the startvec array. By default startvec is initialize with random numbers.
- solveorderconditions: if set to 1, solve the order conditions first before trying to optimize. The default value is 0.
- np: number of processor to use. If np > 1 the MATLAB global optimization toolbox *Multistart* is used. The default value is 1 (just one core).
- max_tries: maximum number of fmincon function calls. The default value is 10.
- writeToFile: whether to write to a file. If set to 1 write the RK coefficients to a file called "ERK-p-s.txt". The default value is 1.

- algorithm: which algorithm to use in fmincon: 'sqp','interior-point', or 'active-set'. By default sqp is used.

Note: numerical experiments have shown that when the objective function is the minimization of the leading truncation error coefficient, the interior-point algorithm performs much better than the sqp one.

- display: level of display of fmincon solver ('off', 'iter', 'notify' or 'final'). The default value is 'notify'.
- problem_class: class of problems for which the RK is designed ('linear' or 'nonlinear' problems). This option changes the type of order conditions check, i.e. linear or nonlinear order conditions controll. The default value is 'nonlinear'.

Note: Only s , p , class and objective are required inputs. All the other arguments are **parameter name - value arguments to the input parser scheme**. Therefore they can be specified in any order.

Example:

```
>> rk=rk_opt(4,3,'erk','acc','max_tries',2,'np',1,'solveorderconditions',1)
```

The fmincon options are set through the optimset that creates/alters optimization options structure. By default the following ad

- MaxFunEvals = 1000000
- TolCon = 1.e-13
- TolFun = 1.e-13
- TolX = 1.e-13
- MaxIter = 10000
- Diagnostics = off
- DerivativeCheck = off
- GradObj = on, if the objective is set equal to 'ssp'

2.1.11 set_n

```
function n=set_n(s,class)
```

Set total number of decision variables

2.1.12 shuosher2butcher

```
function [A,b,c]=shuosher2butcher(alpha,beta);
```

Generate Butcher form of a Runge-Kutta method, given its Shu-Osher or modified Shu-Osher form.

For an m-stage method, α and β should be matrices of dimension $(m+1) \times m$.

2.1.13 test_SSP

```
function test_suite = test_SSP
```

A set of verification tests for the RK-opt package. Currently this tests SSP coefficient optimization and accuracy optimization, but not under constraints on the stability polynomial.

2.1.14 unpack_lsrk

```
function [A,b,bhat,c,alpha,beta,gamma1,gamma2,gamma3,delta]=unpack_lsrk(X,s,class)
```

Extracts the coefficient arrays from the optimization vector.

This function also returns the low-storage coefficients.

2.1.15 unpack_msrk

```
function [A,Ahat,b,bhat,D,theta] =  unpack_msrk(X,s,k,class)
```

Extract the coefficient arrays from the optimization vector

2.1.16 unpack_rk

```
function [A,b,c]=unpack_rk(X,s,class)
```

Extracts the coefficient arrays from the optimization vector.

The coefficients are tored in a single vector x as:

```
x=[A b' c']
```

A is stored row-by-row.

2.1.17 writeField

```
function wf=writeField(writeFid,name,value)
```

2.2 am_radius-opt

Find stability functions with optimal radius of absolute monotonicity. This includes codes for optimizing stability functions of multistep, multistage methods and even methods with downwinding.

Generally, the optimization problem is phrased as a sequence of linear programming feasibility problems. For details, see [ketcheson2009].

The optimization of rational functions is experimental.

Contents

2.2.1 multi_R_opt

```
function multi_R = multi_R_opt(k,p,class,varargin)
```

This function is a script to run the routines Rskp, Rkp_dw, Rkp_imp, or Rkp_imp_dw several times with different inputs, in order to construct tables of optimal values like those that appear in [ketcheson2009]. different values of the input parameters, i.e.:

k = [k1, k2, ..., kK]^T, K = length(k), ith-element = # of steps p = [p1, p2, ..., pP]^T, P = length(p), ith-element = order of accuracy

and

s = [s1, s2, ..., sS]^T, S = length(s), ith-element = # of stages

when optimal contractive k-step, s-stage GLM are investigated.

The family of method to be considered is specified in the string 'class'.

Note that in general $S \neq K \neq P$. Fixed the order of accuracy of the time integration scheme, one is usually interested in understanding the behavior of the threshold factor R as a function of the number of stages. Therefore, for a fixed element of the array "p", this function loops over the elements of the array "s". Thus, min(s) => max(p). The equality holds for any order of accuracy because the number of linear order conditions that will be imposed to construct the GLM coefficients is p.

2.2.2 radimpfast

```
function rad=radimpfast(p,q)
```

Compute the radius of absolute monotonicity of a rational function.

This function is outdated and needs to be fixed.

Uses van de Griend's algorithm, assuming multiplicity one for all roots. Uses high precision arithmetic.

2.2.3 Rkp

```
function [R,alpha,beta]=Rkp(k,p)
```

Find the optimal SSP k-step explicit LMM with order of accuracy p.

Inputs:

- k = # of steps

- p = order of accuracy

Outputs:

- α, β = the coefficients of the method

Requires MATLAB's optimization toolbox for the LP solver.

2.2.4 Rkp_dw

```
function [R,alpha,beta,tbeta]=Rkp_dw(k,p)
```

Finds the optimal SSP k-step explicit LMM with order of accuracy p allowing downwind operators

Inputs: k = # of steps p = order of accuracy

Outputs: alpha, beta, tbeta = the coefficients of the method

The method is given by $u_n = \sum_{j=0}^{k-1}(\alpha[j] + \beta[j]F(u_{n-k+j} + tbeta[j]tF(u_{n-k+j}))$ where tF(u) is the negated downwind operator.

Depends on MATLAB's optimization toolbox for the LP solver

2.2.5 Rkp_imp

```
function [R,alpha,beta]=Rkp_imp(k,p)
```

Find the optimal SSP k-step implicit LMM with order of accuracy p

Inputs:

- k = # of steps
- p = order of accuracy

Outputs: alpha, beta = the coefficients of the method

Depends on MATLAB's optimization toolbox for the LP solver

2.2.6 Rkp_imp_dw

```
function [R,alpha,beta]=Rkp_imp_dw(k,p)
```

Finds the optimal k-step implicit LMM with order of accuracy p allowing downwinding

Inputs: k = # of steps p = order of accuracy

Outputs: alpha, beta, tbeta = the coefficients of the method

Depends on MATLAB's optimization toolbox for the LP solver

2.2.7 Rskp

```
function [R,gamma]=Rskp(s,k,p)
```

Finds the optimal contractive k-step, s-stage GLM with order of accuracy p for linear problems

Inputs: s = # of stages k = # of steps p = order of accuracy

Outputs: R = threshold factor gamma = coefficients of the polynomials

for k=1, the resulting polynomial is $\sum_{j=0}^{m}(1+z/R)^j$

in general, the resulting stability function is (Fill in)

epends on MATLAB's optimization toolbox for the LP solver

2.3 polyopt

Given a spectrum (typically corresponding to a spatial semi-discretization of a PDE), finds an optimal stability polynomial. The polynomial coefficients can then be used as input to RK-coeff-opt to find a corresponding Runge-Kutta method.

This is the implementation of the algorithm described in [ketcheson-ahmadia]. The code was written by Aron Ahmadia and David Ketcheson.

Contents

2.3.1 opt_poly_bisect

```
function [h,poly_coeff] = opt_poly_bisect(lam,s,p,basis,varargin)
```

Finds an optimally stable polynomial of degree s and order p for the spectrum lam in the interval (h_min,h_max) to precision eps.

Optional arguments:

lam_func: A function used to generate the appropriate spectrum at each bisection step, instead of using a fixed (scaled) spectrum. Used for instance to find the longest rectangle of a fixed height (see Figure 10 of the CAMCoS paper).

Examples:

- To find negative real axis inclusion:

```
lam = spectrum('realaxis',500);
s = 10; p = 2;
[h,poly_coeff] = opt_poly_bisect(lam,s,p,'chebyshev')
```

- To reproduce figure 10 of [ketcheson-ahmadia]

```
lam_func = @(kappa) spectrum('rectangle',100,kappa,10)
[h,poly_coeff] = opt_poly_bisect(lam,20,1,'chebyshev','lam_func',lam_func)
plotstabreg_func(poly_coeff,[1])
```

2.3.2 spectrum

```
function lamda = spectrum(name,N,kappa,beta)
```

Return N discretely sampled values from certain sets in the complex plane.

Acceptable values for name:

- 'realaxis': $[-1, 0]$
- 'imagaxis': $[-i, i]$
- 'disk': $z : |z + 1| = 1$
- 'rectangle': $x + iy : -\beta \leq y \leq \beta, -\kappa \leq x \leq 0$
- 'Niegemann-ellipse' and 'Niegemann-circle': See Niegemann 2011
- 'gap': Spectrum with a gap; see Ketcheson & Ahmadia 2012

kappa and beta are used only if name == 'rectangle'

2.3.3 test_polyopt

```
function test_suite = test_polyopt
```

A set of verification tests for the polyopt suite.

2.4 RKtools

Some general utilities for analyzing Runge-Kutta methods.

Contents

2.4.1 am_radius

```
function r = am_radius(A,b,c,eps,rmax)
```

Evaluates the Radius of absolute monotonicity of a Runge-Kutta method, given the Butcher array.

For an m-stage method, A should be an $m \times m$ matrix and b should be a column vector of length m.

Accuracy can be changed by modifying the value of eps (default 10^-10) Methods with very large radii of a.m. (>50) will require the default value of rmax to be increased.

The radius of absolute monotonicity is the largest value of r such that

$$K(I + rA)^{-1} \geq 0 \tag{2.1}$$
$$rK(I + rA)^{-1}e_m \leq e_{m+1} \tag{2.2}$$

where

$$K = \begin{pmatrix} A \\ b^T \end{pmatrix}$$

2.4.2 internal_stab_explicit_butcher

```
function [stability] = internal_stab_explicit_butcher(A,b,c,spectrum,one_step_dt,p)
```

This function computes and plots both intermediate and one-step internal stability vector of an explicit Runge-Kutta scheme given its Butcher tableau.

Note that for an explicit Runge-Kutta scheme the stability functions are polynomials in the complex variable z.

Construct the intermediate stability functions psi_j (where j is the index of the stage).

Note that for an explicit scheme the intermediate stability polynomial associated to the first stage is always 1, i.e. psi_1 = 1. Therefore we just compute and plot the remaining (s-1) intermediate stability polynomials plus the one-step stability polynomial of the Runge-Kuatta method.

2.4.3 L2_timestep_poly

```
function c = L2_timestep_poly(sdisc,p,q,eps,tol)
```

Find the absolutely timestep for a given combination of linear spatial discretization and stability function.

Also (optionally) plots the region of absolute stability and the eigenvalues.

The timestep is determined to within accuracy eps (default 10^-4).

The spectral stability condition is checked to within tol (default 10^-13).

2.4.4 optimal_shuosher_form

```
function [v,alpha,beta] = optimal_shuosher_form(A,b,c)
```

2.4.5 plotstabreg(rk,plotbounds,ls,lw)

```
function plotstabreg(rk,plotbounds,ls,lw)
```

Plots the absolute stability region of a Runge-Kutta method, given the Butcher array

2.4.6 plotstabreg_func

```
function [dummy] = plotstabreg_func(p,q,bounds,ls,lw)
```

plot the absolute stability region of a one-step method, given the stability function

Inputs:

- p: coefficients of the numerator of the stability function
- q: coefficients of the denominator of the stability function

if q is omitted, it is assumed that the function is a polynomial

Remaining inputs are optional:

- bounds: bounds for region to compute and plot (default [-9 1 -5 5])
- ls: line style (default '-r')
- lw: line width (default 2)

2.4.7 rk_stabfun

```
function [p,q] = rk_stabfun(rk)
```

Outputs the stability function of a RK method. The Butcher coefficients should be stored in rk.A, rk.b.

p contains the coefficients of the numerator

q contains the coefficients of the denominator

$$\phi(z) = \frac{\sum_j p_j z^j}{\sum_j q_j z^j} = \frac{\det(I - z(A + eb^T))}{\det(I - zA)}.$$

2.4.8 semispectrum

```
function L = semispectrum(method,order,doplot,nx,cfl)
```

Plot spectra of various semi-discretizations of the advection equation

Current choices for method:

- 'fourier': Fourier spectral method
- 'chebyshev': Chebyshev spectral method
- 'updiff': Upwind difference operators (linearized WENO)
- 'DG': Discontinuous Galerkin method

The value of order matters only for the 'updiff' and 'DG' methods and selects the order of accuracy in those cases.

CHAPTER

THREE

CONTRIBUTING

If you wish to contribute, we recommend that you fork the RK-Opt GitHub repository, implement your additions, and issue a pull request. You may also simply e-mail a patch to us.

CHAPTER

FOUR

BIBLIOGRAPHY

The papers listed here were produced using RK-Opt, and several of them describe parts of it. RK-Opt can be used to easily reproduce many of the results in these papers.

[parsaniRK] Parsani M, Ketcheson DI, Deconinck W. Optimized explicit Runge-Kutta schemes for the spectral difference method applied to wave propagation problems. SIAM Journal on Scientific Computing. 2013. In press; preprint available at: http://arxiv.org/abs/1207.5830 (also uses polyopt)

[ketcheson2011a] Ketcheson DI. Step Sizes for Strong Stability Preservation with Downwind-Biased Operators. SIAM Journal on Numerical Analysis. 2011;49(4):1649. Available at: http://numerics.kaust.edu.sa/papers/dwrk2011/downwind_ssp.html (also uses am_radius-opt)

[ketcheson2011b] Ketcheson DI, Gottlieb S, Macdonald CB. Strong Stability Preserving Two-step Runge–Kutta Methods. SIAM Journal on Numerical Analysis. 2011;49(6):2618.

[ketcheson2010] Ketcheson DI. Runge–Kutta methods with minimum storage implementations. Journal of Computational Physics. 2010;229(5):1763–1773. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0021999109006251

[ketcheson2009b] Ketcheson DI, Macdonald CB, Gottlieb S. Optimal implicit strong stability preserving Runge–Kutta methods. Applied Numerical Mathematics. 2009;59(2):373–392. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0168927408000688

[ketcheson2008] Ketcheson DI. Highly Efficient Strong Stability Preserving Runge-Kutta Methods with Low-Storage Implementations. SIAM Journal on Scientific Computing. 2008;30:2113–2136.

[ketcheson2009] Ketcheson, David I. 2009. "Computation of optimal monotonicity preserving general linear methods." Mathematics of Computation 78(267): 1497–1513.

[ketcheson-ahmadia] Ketcheson DI, Ahmadia AJ. Optimal stability polynomials for numerical integration of initial value problems. Communications in Applied Mathematics and Computational Science. 2012;7(2):247–271. Available for free at: http://numerics.kaust.edu.sa/papers/stability_polynomials/stability_polynomials_2012.html