rdex-loopr

Claude Heiland-Allen

2018–2019

Contents

1 .gitignore 2
2 LICENSE.md 2
3 Makefile 6
4 rdex-loopr.c 7
5 README.md 16

1 .gitignore

```
rdex-loopr
-
```

2 LICENSE.md

```
# Free Art License 1.3 (FAL 1.3)

## Preamble

The Free Art License grants the right to freely copy, distribute, and
transform creative works without infringing the author's rights.

The Free Art License recognizes and protects these rights. Their
implementation has been reformulated in order to allow everyone to use
creations of the human mind in a creative manner, regardless of their
types and ways of expression.

While the public's access to creations of the human mind usually is
restricted by the implementation of copyright law, it is favoured by the
Free Art License. This license intends to allow the use of a work's
resources; to establish new conditions for creating in order to increase
creation opportunities. The Free Art License grants the right to use a
work, and acknowledges the right holder's and the user's rights and
responsibility.

The invention and development of digital technologies, Internet and Free
Software have changed creation methods: creations of the human mind can
obviously be distributed, exchanged, and transformed. They allow to
produce common works to which everyone can contribute to the benefit of
all.

The main rationale for this Free Art License is to promote and protect
these creations of the human mind according to the principles of
copyleft: freedom to use, copy, distribute, transform, and prohibition
of exclusive appropriation.
```

Definitions

work either means the initial work, the subsequent works or the common work as defined hereafter:

common work means a work composed of the initial work and all subsequent contributions to it (originals and copies). The initial author is the one who, by choosing this license, defines the conditions under which contributions are made.

Initial work means the work created by the initiator of the common work (as defined above), the copies of which can be modified by whoever wants to

Subsequent works means the contributions made by authors who participate in the evolution of the common work by exercising the rights to reproduce, distribute, and modify that are granted by the license.

Originals (sources or resources of the work) means all copies of either the initial work or any subsequent work mentioning a date and used by their author(s) as references for any subsequent updates, interpretations, copies or reproductions.

Copy means any reproduction of an original as defined by this license.

1. OBJECT

The aim of this license is to define the conditions under which one can use this work freely.

2. SCOPE

This work is subject to copyright law. Through this license its author specifies the extent to which you can copy, distribute, and modify it.

2.1 FREEDOM TO COPY (OR TO MAKE REPRODUCTIONS)

You have the right to copy this work for yourself, your friends or any other person, whatever the technique used.

2.2 FREEDOM TO DISTRIBUTE, TO PERFORM IN PUBLIC

You have the right to distribute copies of this work; whether modified or not, whatever the medium and the place, with or without any charge, provided that you:

- attach this license without any modification to the copies of this work or indicate precisely where the license can be found,
- specify to the recipient the names of the author(s) of the originals, including yours if you have modified the work,
- specify to the recipient where to access the originals (either initial or subsequent).

The authors of the originals may, if they wish to, give you the right to distribute the originals under the same conditions as the copies.

2.3 FREEDOM TO MODIFY

You have the right to modify copies of the originals (whether initial or subsequent) provided you comply with the following conditions:

- all conditions in article 2.2 above, if you distribute modified copies;
- indicate that the work has been modified and, if it is possible, what kind of modifications have been made;
- distribute the subsequent work under the same license or any compatible license.

The author(s) of the original work may give you the right to modify it under the same conditions as the copies.

3. RELATED RIGHTS

Activities giving rise to author's rights and related rights shall not challenge the rights granted by this license.
For example, this is the reason why performances must be subject to the same license or a compatible license. Similarly, integrating the work in a database, a compilation or an anthology shall not prevent anyone from using the work under the same conditions as those defined in this license.

4. INCORPORATION OF THE WORK

Incorporating this work into a larger work that is not subject to the Free Art License shall not challenge the rights granted by this license.
If the work can no longer be accessed apart from the larger work in which it is incorporated, then incorporation shall only be allowed under the condition that the larger work is subject either to the Free Art License or a compatible license.

5. COMPATIBILITY

A license is compatible with the Free Art License provided:

- it gives the right to copy, distribute, and modify copies of the work including for commercial purposes and without any other restrictions than those required by the respect of the other compatibility criteria;
- it ensures proper attribution of the work to its authors and access to previous versions of the work when possible;
- it recognizes the Free Art License as compatible (reciprocity);
- it requires that changes made to the work be subject to the same license or to a license which also meets these compatibility criteria.

6. YOUR INTELLECTUAL RIGHTS

This license does not aim at denying your author's rights in your contribution or any related right. By choosing to contribute to the development of this common work, you only agree to grant others the same rights with regard to your contribution as those you were granted by this license. Conferring these rights does not mean you have to give up your intellectual rights.

7. YOUR RESPONSIBILITIES

The freedom to use the work as defined by the Free Art License (right to copy, distribute, modify) implies that everyone is responsible for their own actions.

8. DURATION OF THE LICENSE

This license takes effect as of your acceptance of its terms. The act of copying, distributing, or modifying the work constitutes a tacit agreement. This license will remain in effect for as long as the copyright which is attached to the work. If you do not respect the terms of this license, you automatically lose the rights that it confers.
If the legal status or legislation to which you are subject makes it impossible for you to respect the terms of this license, you may not make use of the rights which it confers.

9. VARIOUS VERSIONS OF THE LICENSE

This license may undergo periodic modifications to incorporate improvements by its authors (instigators of the Copyleft Attitude movement) by way of new, numbered versions.
You will always have the choice of accepting the terms contained in the version under which the copy of the work was distributed to you, or alternatively, to use the provisions of one of the subsequent versions.

10. SUB-LICENSING

Sub-licenses are not authorized by this license. Any person wishing to make use of the rights that it confers will be directly bound to the authors of the common work.

11. LEGAL FRAMEWORK

This license is written with respect to both French law and the Berne Convention for the Protection of Literary and Artistic Works.

USER GUIDE

How to use the Free Art License?

To benefit from the Free Art License, you only need to mention the following elements on your work:

> [Name of the author, title, date of the work. When applicable, names of authors of the common work and, if possible, where to find the originals].
>
> Copyleft: This is a free work, you can copy, distribute, and modify it under the terms of the Free Art License
> <http://artlibre.org/licence/lal/en/>

Why to use the Free Art License?

1. To give the greatest number of people access to your work.
2. To allow it to be distributed freely.

```
3. To allow it to evolve by allowing its copy, distribution, and
   transformation by others.
4. So that you benefit from the resources of a work when it is under the
   Free Art License: to be able to copy, distribute or transform it
   freely.
5. But also, because the Free Art License offers a legal framework to
   disallow any misappropriation. It is forbidden to take hold of your work
   and bypass the creative process for one's exclusive possession.

### When to use the Free Art License?

Any time you want to benefit and make others benefit from the right to
copy, distribute and transform creative works without any exclusive
appropriation, you should use the Free Art License. You can for example
use it for scientific, artistic or educational projects.

### What kinds of works can be subject to the Free Art License?

The Free Art License can be applied to digital as well as physical
works.
You can choose to apply the Free Art License on any text, picture,
sound, gesture, or whatever sort of stuff on which you have sufficient
author's rights.

### Historical background of this license:

It is the result of observing, using and creating digital technologies,
free software, the Internet and art. It arose from the Copyleft
Attitude meetings which took place in Paris in 2000. For the first
time, these meetings brought together members of the Free Software
community, artists, and members of the art world. The goal was to adapt
the principles of Copyleft and free software to all sorts of creations.

------------------------------------------------------------------------

<http://www.artlibre.org>
Copyleft Attitude, 2007.

You can make reproductions and distribute this license verbatim
(without any changes).

Translation: Jonathan Clarke, Benjamin Jean, Griselda Jung, Fanny Mourguet,
Antoine Pitrou. Thanks to <http://framalang.org>
```

3 Makefile

```
all: rdex-loopr

clean:
	-rm rdex-loopr

rdex-loopr: rdex-loopr.c
	gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native -o rdex-loopr ↙
	    ↳ rdex-loopr.c `pkg-config gsl --libs`

.PHONY: all clean
```

4 rdex-loopr.c

```
#define _POSIX_C_SOURCE 2

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include <gsl/gsl_multiroots.h>

typedef double real;

// S is image size
// time taken is O(S^4)
// S = 24 : 4 minutes (conjectured)
// S = 32 : 11 minutess
// S = 48 : 55 minutes
// S = 64 : 3 hours (conjectured)
#define S 32

// Q is preperiod
// P0 is maximum period
// time taken is O(Q+P0)
#define Q 1
#define Q0 20000
#define P0 2000

// derived dimensions
#define W S
#define H S
#define D (2 * W * H)

// image buffer
unsigned char ppm[H][W][3];

// vector indexes (wrap-around)
#define U(j,i) (((((j)+H)%H)*W + (((i)+W)%W) * 2 + 0)
#define V(j,i) (((((j)+H)%H)*W + (((i)+W)%W) * 2 + 1)

// parameters for reaction diffusion
struct rdex
{
  real Du, Dv, F, k, dt;
  int P;
};

void rdex_variance(const gsl_vector *src, real *su, real *sv, real *u1, real *v1↙
    ↳ )
{
  // compute variance
  real su0 = D / 2, su1 = 0, su2 = 0
     , sv0 = D / 2, sv1 = 0, sv2 = 0;
  for (int i = 0; i < D / 2; ++i)
  {
    real u = gsl_vector_get(src, 2 * i + 0);
    real v = gsl_vector_get(src, 2 * i + 1);
```

```
      su1 += u;
      sv1 += v;
      su2 += u * u;
      sv2 += v * v;
    }
    *su = (su0 * su2 - su1 * su1) / (su0 * su0);
    *sv = (sv0 * sv2 - sv1 * sv1) / (sv0 * sv0);
    *u1 = su1;
    *v1 = sv1;
}

// one step reaction diffusion
void rdex_f1(struct rdex *r, gsl_vector *dst, gsl_vector *src)
{
  const real Du = r->Du;
  const real Dv = r->Dv;
  const real F  = r->F;
  const real k  = r->k;
  const real dt = r->dt;
  const real w[3] = { -(4 + 4/sqrt(2)), 1, 1/sqrt(2) };
  for (int j = 0; j < H; ++j)
  {
    for (int i = 0; i < W; ++i)
    {
      real u = gsl_vector_get(src, U(j, i));
      real v = gsl_vector_get(src, V(j, i));
      real uvv = u * v * v;
      real Lu = 0;
      real Lv = 0;
      for (int dj = -1; dj <= 1; ++dj)
      {
        for (int di = -1; di <= 1; ++di)
        {
          int k = abs(dj) + abs(di);
          const real u1 = gsl_vector_get(src, U(j + dj, i + di));
          const real v1 = gsl_vector_get(src, V(j + dj, i + di));
          Lu += w[k] * u1;
          Lv += w[k] * v1;
        }
      }
      real du = Du * Lu - uvv + F * (1 - u);
      real dv = Dv * Lv + uvv - (F + k) * v;
      u += du * dt;
      v += dv * dt;
      u = fmin(fmax(u, 0), 1);
      v = fmin(fmax(v, 0), 1);
      gsl_vector_set(dst, U(j, i), u);
      gsl_vector_set(dst, V(j, i), v);
    }
  }
}

void rdex_fdf1(struct rdex *r, gsl_vector *dst, gsl_vector *src, gsl_matrix *↙
    ↳ dstJ, gsl_matrix *srcJ)
{
  const real Du = r->Du;
  const real Dv = r->Dv;
```

```
  const real F  = r->F;
  const real k  = r->k;
  const real dt = r->dt;
  const real w[3] = { -(4 + 4/sqrt(2)), 1, 1/sqrt(2) };
  for (int j = 0; j < D; ++j)
    for (int i = 0; i < D; ++i)
      gsl_matrix_set(dstJ, j, i, 0);
  for (int j = 0; j < H; ++j)
  {
    for (int i = 0; i < W; ++i)
    {
      real u = gsl_vector_get(src, U(j, i));
      real v = gsl_vector_get(src, V(j, i));
      real uvv = u * v * v;
      real Lu = 0;
      real Lv = 0;
      for (int dj = -1; dj <= 1; ++dj)
      {
        for (int di = -1; di <= 1; ++di)
        {
          int k = abs(dj) + abs(di);
          const real u1 = gsl_vector_get(src, U(j + dj, i + di));
          const real v1 = gsl_vector_get(src, V(j + dj, i + di));
          Lu += w[k] * u1;
          Lv += w[k] * v1;
          gsl_matrix_set(dstJ, U(j, i), U(j + dj, i + di), gsl_matrix_get(dstJ, U(j, i), U(j + dj, i + di))
            + Du * w[k] * dt * gsl_matrix_get(srcJ, U(j + dj, i + di), U(j + dj, i + di)));
          gsl_matrix_set(dstJ, V(j, i), V(j + dj, i + di), gsl_matrix_get(dstJ, V(j, i), V(j + dj, i + di))
            + Dv * w[k] * dt * gsl_matrix_get(srcJ, V(j + dj, i + di), V(j + dj, i + di)));
        }
      }
      gsl_matrix_set(dstJ, U(j, i), U(j, i), gsl_matrix_get(dstJ, U(j, i), U(j, i))
        + gsl_matrix_get(srcJ, U(j, i), U(j, i))
        - dt * (v * v + F) * gsl_matrix_get(srcJ, U(j, i), U(j, i))
        - dt * 2 * u * v * gsl_matrix_get(srcJ, V(j, i), U(j, i)));
      gsl_matrix_set(dstJ, U(j, i), V(j, i), gsl_matrix_get(dstJ, U(j, i), V(j, i))
        - dt * 2 * u * v * gsl_matrix_get(srcJ, V(j, i), V(j, i))
        - dt * v * v * gsl_matrix_get(srcJ, U(j, i), V(j, i)));
      gsl_matrix_set(dstJ, V(j, i), U(j, i), gsl_matrix_get(dstJ, V(j, i), U(j, i))
        + dt * v * v * gsl_matrix_get(srcJ, U(j, i), U(j, i))
        + dt * 2 * u * v * gsl_matrix_get(srcJ, V(j, i), V(j, i)));
      gsl_matrix_set(dstJ, V(j, i), V(j, i), gsl_matrix_get(dstJ, V(j, i), V(j, i))
        + gsl_matrix_get(srcJ, V(j, i), V(j, i))
        + dt * v * v * gsl_matrix_get(srcJ, U(j, i), V(j, i))
        + dt * (2  * u * v - (F + k)) * gsl_matrix_get(srcJ, V(j, i), V(j, i)));
      real du = Du * Lu - uvv + F * (1 - u);
      real dv = Dv * Lv + uvv - (F + k) * v;
      u += du * dt;
      v += dv * dt;
```

```
        u = fmin(fmax(u, 0), 1);
        v = fmin(fmax(v, 0), 1);
        gsl_vector_set(dst, U(j, i), u);
        gsl_vector_set(dst, V(j, i), v);
      }
    }
  }

  int rdex_f(const gsl_vector *x, void *p, gsl_vector *y)
  {
    struct rdex *r = (struct rdex *) p;
    gsl_vector *src = gsl_vector_alloc(D);
    gsl_vector *dst = gsl_vector_alloc(D);
    gsl_vector *mid = gsl_vector_alloc(D);
    gsl_vector *top = gsl_vector_alloc(D);
    for (int i = 0; i < D; ++i)
    {
      gsl_vector_set(src, i, gsl_vector_get(x, i));
    }
    real best_sum = 1.0 / 0.0;
    for (int n = 0; n < Q + P0; ++n)
    {
      if (n == Q)
      {
        for (int i = 0; i < D; ++i)
        {
          gsl_vector_set(mid, i, gsl_vector_get(src, i));
        }
      }
      rdex_f1(r, dst, src);
      gsl_vector *t = src; src = dst; dst = t;
      if (n >= Q + 100)
      {
        real sum = 0;
        for (int i = 0; i < D; ++i)
        {
          real d = gsl_vector_get(src, i) - gsl_vector_get(mid, i);
          sum += d * d;
        }
        if (sum < best_sum)
        {
          best_sum = sum;
          r->P = n + 1 - Q;
          for (int i = 0; i < D; ++i)
          {
            gsl_vector_set(top, i, gsl_vector_get(src, i));
          }
        }
      }
    }
    gsl_vector *t = src; src = top; top = t;
    // compute variance
    real su, sv;
    real dummy;
    rdex_variance(src, &su, &sv, &dummy, &dummy);
    real s = su + sv;
    // divide by variance to avoid convergence to featureless images
```

```
  for (int i = 0; i < D / 2; ++i)
  {
    real u0 = gsl_vector_get(mid, 2 * i + 0);
    real v0 = gsl_vector_get(mid, 2 * i + 1);
    real u1 = gsl_vector_get(src, 2 * i + 0);
    real v1 = gsl_vector_get(src, 2 * i + 1);
    gsl_vector_set(y, 2 * i + 0, (u1 - u0) / s);
    gsl_vector_set(y, 2 * i + 1, (v1 - v0) / s);
  }
  gsl_vector_free(src);
  gsl_vector_free(dst);
  gsl_vector_free(mid);
  gsl_vector_free(top);
  return GSL_SUCCESS;
}

int rdex_fdf(const gsl_vector *x, void *p, gsl_vector *y, gsl_matrix *J)
{
  struct rdex *r = (struct rdex *) p;
  gsl_vector *src = gsl_vector_alloc(D);
  gsl_vector *dst = gsl_vector_alloc(D);
  gsl_vector *mid = gsl_vector_alloc(D);
  gsl_vector *top = gsl_vector_alloc(D);
  gsl_matrix *srcJ = gsl_matrix_alloc(D, D);
  gsl_matrix *dstJ = gsl_matrix_alloc(D, D);
  gsl_matrix *midJ = gsl_matrix_alloc(D, D);
  gsl_matrix *topJ = gsl_matrix_alloc(D, D);
  for (int i = 0; i < D; ++i)
  {
    gsl_vector_set(src, i, gsl_vector_get(x, i));
    for (int j = 0; j < D; ++j)
      gsl_matrix_set(srcJ, i, j, i == j);
  }
  real best_sum = 1.0 / 0.0;
  for (int n = 0; n < Q + P0; ++n)
  {
    if (n == Q)
    {
      for (int i = 0; i < D; ++i)
      {
        gsl_vector_set(mid, i, gsl_vector_get(src, i));
        for (int j = 0; j < D; ++j)
          gsl_matrix_set(midJ, i, j, gsl_matrix_get(srcJ, i, j));
      }
    }
    rdex_fdf1(r, dst, src, dstJ, srcJ);
    gsl_vector *t = src; src = dst; dst = t;
    gsl_matrix *tJ = srcJ; srcJ = dstJ; dstJ = tJ;
    if (n >= Q + 100)
    {
      real sum = 0;
      for (int i = 0; i < D; ++i)
      {
        real d = gsl_vector_get(src, i) - gsl_vector_get(mid, i);
        sum += d * d;
      }
      if (sum < best_sum)
```

```
      {
        best_sum = sum;
        r->P = n + 1 - Q;
        for (int i = 0; i < D; ++i)
        {
          gsl_vector_set(top, i, gsl_vector_get(src, i));
          for (int j = 0; j < D; ++j)
            gsl_matrix_set(topJ, i, j, gsl_matrix_get(srcJ, i, j));
        }
      }
    }
  }
  gsl_vector *t = src; src = top; top = t;
  gsl_matrix *tJ = srcJ; srcJ = topJ; topJ = tJ;
  real su, sv, su1, sv1;
  rdex_variance(src, &su, &sv, &su1, &sv1);
  real s = su + sv;
  // divide by variance to avoid convergence to featureless images
  for (int i = 0; i < D / 2; ++i)
  {
    real u0 = gsl_vector_get(mid, 2 * i + 0);
    real v0 = gsl_vector_get(mid, 2 * i + 1);
    real u1 = gsl_vector_get(src, 2 * i + 0);
    real v1 = gsl_vector_get(src, 2 * i + 1);
    gsl_vector_set(y, 2 * i + 0, (u1 - u0) / s);
    gsl_vector_set(y, 2 * i + 1, (v1 - v0) / s);
  }
  real s0 = D / 2;
  real s1[2] = { su1, sv1 };
  // J( (u1 - u0) / s ) = J(u1 - u0) / s - ((u1 - u0) J(s)) / s^2
  for (int i = 0; i < D; ++i)
    for (int j = 0; j < D; ++j)
    {
      gsl_matrix_set(J, i, j,
        (gsl_matrix_get(srcJ, i, j) - gsl_matrix_get(midJ, i, j)) / s -
        (gsl_vector_get(src, i) - gsl_vector_get(mid, i)) *
          (2 * gsl_vector_get(src, j) / s0 - 2 * s1[j&1] / (s0 * s0)) / (s * s)
        );
    }
  gsl_vector_free(src);
  gsl_vector_free(dst);
  gsl_vector_free(mid);
  gsl_vector_free(top);
  gsl_matrix_free(srcJ);
  gsl_matrix_free(dstJ);
  gsl_matrix_free(midJ);
  gsl_matrix_free(topJ);
  return GSL_SUCCESS;
}

void output(gsl_vector *src, gsl_vector *dst, int seed, int try, int iter, ↙
    ↳ struct rdex *params, real norm)
{
        char command[1000];
        snprintf(command, 1000, "ffmpeg -loglevel quiet -framerate 60 -i - '%08X↙
            ↳ -%08d-%08d-%08d-%.8e.gif'", seed, try, iter, params->P, norm);
        fprintf(stderr, "%08X\t%d\t%d\t%d\t%.8e\tGIF\n", seed, try, iter, params↙
```

```
    ↪ ->P, norm);
  FILE *file = popen(command, "w");
  int last_frame = 0;
  real frame = 0.5;
  for (int n = 0; n < Q + params->P; ++n)
  {
    if (Q <= n)
    {
      frame += 256.0 / params->P;
      if (frame >= last_frame + 1)
      {
        for (int j = 0; j < H; ++j)
        {
          for (int i = 0; i < W; ++i)
          {
            real u = 1 - gsl_vector_get(src, U(j, i));
            real v = gsl_vector_get(src, V(j, i));
            real h = 16 * atan2(v, u);
            real l = fmax(0, cos(16 * (v * v + u * u)));
            real c1 = 0.5 * cos(h);
            real c2 = 0.5 * sin(h);
            real r = l + 1.407 * c1;
            real g = l - 0.677 * c1 - 0.236 * c2;
            real b = l               + 1.848 * c2;
            ppm[j][i][0] = 255 * fmin(fmax(r, 0), 1);
            ppm[j][i][1] = 255 * fmin(fmax(g, 0), 1);
            ppm[j][i][2] = 255 * fmin(fmax(b, 0), 1);
          }
        }
        fprintf(file, "P6\n%d %d\n255\n", W, H);
        fwrite(&ppm[0][0][0], H * W * 3, 1, file);
        fflush(file);
        last_frame += 1;
      }
    }
    rdex_f1(params, dst, src);
    gsl_vector *t = src; src = dst; dst = t;
  }
  pclose(file);
  gsl_vector *t = src; src = dst; dst = t;
  snprintf(command, 1000, "%08X-%08d-%08d-%08d-%.8e.ppm", seed, try, iter,↙
    ↪ params->P, norm);
  file = fopen(command, "wb");
        for (int j = 0; j < H; ++j)
        {
          for (int i = 0; i < W; ++i)
          {
            real u = 1 - gsl_vector_get(src, U(j, i));
            real v = gsl_vector_get(src, V(j, i));
            real h = 16 * atan2(v, u);
            real l = fmax(0, cos(16 * (v * v + u * u)));
            real c1 = 0.5 * cos(h);
            real c2 = 0.5 * sin(h);
            real r = l + 1.407 * c1;
            real g = l - 0.677 * c1 - 0.236 * c2;
            real b = l               + 1.848 * c2;
            ppm[j][i][0] = 255 * fmin(fmax(r, 0), 1);
```

```
                    ppm[j][i][1] = 255 * fmin(fmax(g, 0), 1);
                    ppm[j][i][2] = 255 * fmin(fmax(b, 0), 1);
                  }
                }
                fprintf(file, "P6\n%d %d\n255\n", W, H);
                fwrite(&ppm[0][0][0], H * W * 3, 1, file);
        fclose(file);
}

// main program entry point
int main(int argc, char **argv)
{
  // seed PRNG with time and argument
  int seed = time(0) + (argc > 1 ? argv[1][0] : 0);
  fprintf(stderr, "seed = %d\n", seed);
  srand(seed);

  // avoid abort() from default error handler when matrix is singular
  gsl_set_error_handler_off();

  // example parameters
#if 0
  struct rdex params = { 0.118022, 0.043133, 0.023135, 0.055081, 0.5 };
  struct rdex params = { 0.020185, 0.013866, 0.016683, 0.050902, 0.5, 0 }; // https://rdex.mathr.co.uk/kiblix/colour/1581 dynamic small
  struct rdex params = { 0.037655, 0.026061, 0.024382, 0.051716, 0.5 }; // https://rdex.mathr.co.uk/kiblix/nearby/155 dynamic medium
  struct rdex params = { 0.070642, 0.043482, 0.024294, 0.052547, 0.5 }; // https://rdex.mathr.co.uk/kiblix/nearby/1842 dynamic medium
  struct rdex params = { 0.157658, 0.089861, 0.019432, 0.047631, 0.5 }; // https://rdex.mathr.co.uk/kiblix/colour/227 dynamic large
  struct rdex params = { 0.140444, 0.051544, 0.093481, 0.064434, 0.5 }; // https://rdex.mathr.co.uk/kiblix/nearby/2082 static lines (green)
  struct rdex params = { 0.255443, 0.090206, 0.044084, 0.060603, 0.5 }; // https://rdex.mathr.co.uk/kiblix/colour/1586 static lines (purple)
#endif
  struct rdex params = { 0.115447, 0.059511, 0.018156, 0.045008, 0.5, 0 };

  // allocate root solver
  const gsl_multiroot_fdfsolver_type *T = gsl_multiroot_fdfsolver_newton;// hybrids;
  gsl_multiroot_fdfsolver *s = gsl_multiroot_fdfsolver_alloc(T, D);
  gsl_multiroot_function_fdf F;
  F.f = &rdex_f;
  F.df = 0;
  F.fdf = &rdex_fdf;
  F.n = D;
  F.params = &params;

  // allocate vectors
  gsl_vector *src = gsl_vector_alloc(D);
  gsl_vector *dst = gsl_vector_alloc(D);
  gsl_vector *tmp = gsl_vector_alloc(D);

  // calculate forever
  int try = 0;
  do
```

```
  {
435
    // initialize with pinkish noise source
    params.P = P0;
    int best_period = P0;
    real best_norm = 1.0 / 0.0;
440 for (int n = 0; n < 1000; ++n)
    {
      for (int i = 0; i < D; ++i)
        gsl_vector_set(src, i, 0.5);
      real noise = 0.5;
445   for (int w = 1, h = 1; w < W && h < H; w <<= 1, h <<= 1, noise /= 2)
        for (int j = 0; j < h; ++j)
        for (int i = 0; i < w; ++i)
        {
          real ru = noise * (rand() / (double) RAND_MAX - 0.5);
450       real rv = noise * (rand() / (double) RAND_MAX - 0.5);
          for (int dj = 0; dj < H/h; ++dj)
          for (int di = 0; di < W/w; ++di)
          {
            int ui = U(j*H/h + dj + H/h/2, i*W/w + di + W/w/2);
455         int vi = V(j*H/h + dj + H/h/2, i*W/w + di + W/w/2);
            gsl_vector_set(src, ui, gsl_vector_get(src, ui) + ru);
            gsl_vector_set(src, vi, gsl_vector_get(src, vi) + rv);
          }
        }
460   // iterate a whole bunch
      for (int m = 0; m < Q0; ++m)
      {
        rdex_f1(&params, dst, src);
        gsl_vector *t = dst; dst = src; src = t;
465   }
      real su, sv, dummy;
      rdex_variance(src, &su, &sv, &dummy, &dummy);
      real s = su + sv;
      if (! (s > 1e-3)) continue;
470   rdex_f(src, &params, dst);
      real norm = 0;
      for (int i = 0; i < D; ++i)
      {
        real f = gsl_vector_get(dst, i);
475     norm += f * f;
      }
      norm /= D;
      norm = sqrt(norm);
      if (norm < best_norm)
480   {
        best_norm = norm;
        best_period = params.P;
        gsl_vector *t = src; src = tmp; tmp = t;
        fprintf(stderr, "%08X\t%d\t%d\t%.18e\n", seed, n, best_period, best_norm↙
            ↳ );
485     if (best_norm < 1)
          break;
      }
    }
    fprintf(stderr, "%08X\t%d\t%d\t%d\t%.18e\t%d\n", seed, try, 0, best_period, ↙
```

```
        ↪ best_norm, 0);
490     params.P = best_period;

        // initialize solver
        gsl_multiroot_fdfsolver_set(s, &F, tmp);

495     // mine for time crystal
        int iter = 0;
        real threshold = 1.0/0.0;
        do
        {
500       iter++;

          // step solver
          int status1 = gsl_multiroot_fdfsolver_iterate(s);

505       // check RMS error
          real norm = 0;
          for (int i = 0; i < D; ++i)
          {
            real f = gsl_vector_get(s->f, i);
510         norm += f * f;
          }
          norm /= D;
          norm = sqrt(norm);
          fprintf(stderr, "%08X\t%d\t%d\t%d\t%.8e\t%d\n", seed, try, iter, params.P,↙
              ↪ norm, status1);
515
          // output if close to cycle
          if (norm < threshold)
          {
            for (int i = 0; i < D; ++i)
520           gsl_vector_set(src, i, gsl_vector_get(s->x, i));
            output(src, dst, seed, try, iter, &params, norm);
            threshold = norm;
          }
          else
525       {
            if (status1 || isnan(norm) || isinf(norm)) break;
          }
        }
        while (iter < 10);
530     try++;
      } while(1);

      // never reached
      gsl_vector_free(src);
535   gsl_vector_free(dst);
      gsl_vector_free(tmp);
      gsl_multiroot_fdfsolver_free(s);
      return 0;
    }
```

5 README.md

```
# rdex-loopr
```

```
Mining for time crystals: repeating patterns in reaction-diffusion systems.

## Prerequisites

GNU Scientific Library, FFMPEG, as well as C build tools.

    sudo apt install libgsl-dev ffmpeg, build-essential

## Compile

    make

## Run

    mkdir output
    cd output
    ../rdex-loopr a &
    ../rdex-loopr b &
    ...

If you have lots of cores and RAM:

    parallel -u ../rdex-loopr ::: A B C D E F G H

## Edit

Change `S` to the desired size.  80 needs 7.5GB RAM, O(S^4).

## Legal

rdex-loopr (C) 2018 Claude Heiland-Allen <claude@mathr.co.uk>

Copyleft: This is a free work, you can copy, distribute, and
modify it under the terms of the Free Art License
<http://artlibre.org/licence/lal/en/>.
```